Theoretical Computer Science 450 (2012) 22–30

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Categorial grammars with iterated types form a strict hierarchy of k-valued languages

Denis Béchet [a,*], Alexandre Dikovsky [a], Annie Foret [b]

[a] LINA UMR CNRS 6241, Université de Nantes, France

[b] IRISA – Université de Rennes1, Campus Universitaire de Beaulieu, Avenue du Général Leclerc, 35042 Rennes Cedex, France

ARTICLE INFO

Keywords:
Formal grammars
Categorial grammars
Language hierarchy
Iterated types

ABSTRACT

The notion of k-valued categorial grammars in which every word is associated to at most k types is often used in the field of lexicalized grammars as a fruitful constraint for obtaining interesting properties like the existence of learning algorithms. This constraint is reasonable only when the classes of k-valued grammars correspond to a real hierarchy of generated languages. Such a hierarchy has been established earlier for the classical categorial grammars.

In this paper the hierarchy by the k-valued constraint is established in the class of categorial grammars extended with iterated types adapted to express the so called projective dependency structures.

1. Introduction

The field of natural language processing includes lexicalized grammars such as classical categorial grammars (AB grammars) [1], the different variants of Lambek calculus [2], lexicalized tree adjoining grammars [3], etc. In these lexicalized formalisms, a k-valued grammar associates at most k categories to each word of the lexicon. For every class of lexicalized grammars this constraint induces the strict hierarchy of grammars for different values of k. As to the hierarchy of the corresponding languages, it might collapse.

In fact, in the field of lexicalized grammars, the concept of k-valued grammars is often used to obtain sub-classes of grammars and languages satisfying an important property which does not hold in the whole class. In particular, this is the way many Gold's model learnability [4] results are obtained. At the same time, this method applies only when the hierarchy of languages is strict. For instance, the strict hierarchy of CF-languages generated by k-valued classical categorial grammars was shown in [5]. In this paper, we prove that the class $*$AB of categorial grammars extended by iterated types also induces a strict hierarchy of languages on k-valued constraints.

In the frame of logical type grammars the iterated types were first introduced in the Categorial Dependency Grammars (CDGs) [6–8] in order to express the optional repeatable dependencies whose existence is postulated as one of the basic principles of dependency syntax (see [9]). For instance, the optional repeatable dependencies of modifiers (adjectives, attributes) on their noun heads and those of adjuncts on their verb heads are common to all languages. Later the iterated types were integrated into pregroup grammars [10]. The grammars in $*$AB considered in this paper are in a sense incomparable with the CDGs because the CDGs do not use the higher order types available in $*$AB, instead they use polarized valencies of first order types, which cannot be expressed in $*$AB and permit to express discontinuous dependencies between the heads and their displaced subordinates (such as fronted WH-junctions or parts of discontinuous comparative

* Corresponding author.
E-mail addresses: Denis.Bechet@univ-nantes.fr (D. Béchet), Alexandre.Dikovsky@univ-nantes.fr (A. Dikovsky), Annie.Foret@irisa.fr (A. Foret).

0304-3975/$ – see front matter
doi:10.1016/j.tcs.2012.04.024

more . . . than constructions in English). The grammars in ∗AB limited to first order types extended by the iteration constitute a subclass of CDGs in which may be expressed continuous (i.e. projective) dependency structures (cf. Example 4 below). At the same time, the repeatable dependencies are a challenge for grammatical inference [11]. For instance, the repeatable circumstantial dependencies A in Example 4 is determined by the unique type $[N\backslash S/A^*]$ assigned to an intransitive verb, and not through consecutive subtypes of iteration-less types $[N\backslash S]$, $[N\backslash S/A]$, $[N\backslash S/A/A]$ . . ..

The main result of this paper shows that the languages generated by k-valued grammars in ∗AB for different k form a strict hierarchy.

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 concludes.

2. Background

2.1. Categorial grammars

The classical categorial grammars is the simplest class of logical type grammars. The basic idea behind their types is that, when a phrase w has a type of the form $B \setminus A$, this means that w can be concatenated on its left with a phrase w_0 of type B, so as to obtain the phrase $w_0 w$ of type A (similar for A / B for the right concatenation). Example 1 below illustrates this principle.

Definition 1 (*Types*)**.** The *types Tp*, or formulae, are generated from a set of *primitive types Pr*, or atomic formulae, by two binary connectives[1] " / " (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 (a set of words);
- $\lambda : \Sigma \mapsto \mathcal{P}^f(Tp)$ is a function (called lexicon) that associates finite subsets of Tp with the words in Σ. We write $G : a \mapsto X$ (or just $a \mapsto X$ when G is implied) if $X \in \lambda(a)$. This means that X is a possible category of a);
- $S \in Pr$ is the *main type* associated with correct sentences.

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*)**.** The language $L(G)$ generated by a categorial grammar G in a class $\mathcal{C}$ is defined through a binary derivation relation $\vdash_{\mathcal{C}}$ on strings of types (i.e. on Tp^*). Traditionally, the derivation relations are defined through type calculi. Given a type calculus for $\mathcal{C}$ and the corresponding derivation relation $\vdash_{\mathcal{C}}$, a sentence $a_1 \ldots a_n$ belongs to $L(G)$, the language of G, if there are associations $a_1 \mapsto X_1, \ldots, a_n \mapsto X_n$ such that $X_1 \ldots X_n \vdash_{\mathcal{C}} S$ ($\mathcal{C}$ will be omitted when implied by the context).

$L_{\mathcal{C}}(G)$ will denote the language of G according to $\vdash_{\mathcal{C}}$.

2.2. ∗AB calculus

Categorial grammars usually express optional and repeatable arguments through recursion. Here, we present a different approach originating from the dependency syntax and formalized through an extended type calculus ∗AB in which 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 naturally expresses the optional repeatable dependencies mentioned in the Introduction.

Definition 4 (*Types*)**.** The *types Tp*, or formulae, 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:

$X / Y, Y \vdash X$	$(\mathbf{L^r})$	$Y, Y \setminus X \vdash X$	$(\mathbf{L^l})$
$X / y^*, y \vdash X / y^*$	$(\mathbf{L^{r*}})$	$y, y^* \setminus X \vdash y^* \setminus X$	$(\mathbf{L^{l*}})$
$X / y^* \vdash X$	$(\Omega^{\mathbf{r}})$	$y^* \setminus X \vdash X$	$(\Omega^{\mathbf{l}})$

The classical AB Calculus consists of the first two elimination rules $\mathbf{L^r}$ and $\mathbf{L^l}$. The corresponding derivation relation is denoted by $\vdash_{AB}$. The AB-grammars are weakly equivalent to the ϵ-free CF-grammars. Indeed, to each ϵ-free Context-Free

[1] No product connective is used in the paper.

Grammar G in *Greibach Normal Form*, one can associate the AB-grammar $cg_{AB}(G)$ with the alphabet consisting of the terminals of G, with the primitive types which are the nonterminals of G, and with the following lexicon:
$a \mapsto ((\ldots(X/X_n)/X_{n-1}\ldots)/X_1)$ for each rule $X \to aX_1\ldots X_{n-1}X_n$ in G.

On the other hand, to each AB grammar G, one can associate the following equivalent CF-grammar $cf(G)$. It has the alphabet of G as terminals, the set $Tp(G)$ of *subformulas of types of G as non-terminals*, and the rules $\{B \to A\ A\backslash B \mid A\backslash B \in Tp(G)\} \cup \{B \to B/A\ A \mid B/A \in Tp(G)\} \cup \{A \to c \mid c \mapsto A \in G\}$.

The equivalence between the two grammars is *weak*, because it concerns only the string languages, not structures.

Example 1. Let $\lambda(John) = \lambda(Mary) = N$ and $\lambda(loves) = [N\backslash S/N]$. Then the sentence *John loves Mary* is generated by both, AB- and $*$AB-grammars. See also Example 4 below, where the iteration rules are involved.

Definition 5 (*Head and Arguments*)**.** Any type X can be written in the following form: $((p|A_1)|\ldots|A_n)$ where $A|B$ stands for A/B or $B\backslash A$ and p is primitive. p is the *head* of X, each subtype $((p|A_1)|\ldots A_k)$ is a *head subtype* of X, n is the *arity* of X, and each A_i is said an *argument subtype* of X.

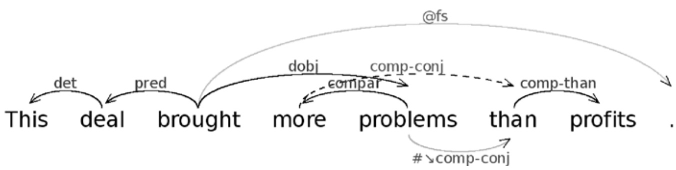
2.3. Categorial Dependency Grammars

As it is mentioned in the Introduction, the Categorial Dependency Grammars (CDGs) [6–8] is an extension of the first-order[2] type subset of $*$AB using so called polarized valencies in order to express discontinuous (non-projective) dependencies. For instance, in Example 2 one can see the discontinuous comparative dependency *comp* − *conj* cut by the projective dependency *dobj* between the main verb and its direct object. To establish this dependency the CDG type of *more* has the positive right valency $\nearrow$*comp* − *conj* and the type of *than* has the dual right negative valency $\searrow$*comp* − *conj*. The CDG-calculus has the rules of $*$AB applied to the first order types with the polarized valencies. The $*$AB rules do not affect the valencies. Besides them CDG-calculus has the following special rule for pairing of dual left polarized valencies (another similar rule pairs the right valencies):

$$\mathbf{D^l}.\ \alpha^{P_1(\swarrow C)P(\nwarrow C)P_2} \vdash \alpha^{P_1 P P_2},$$

if in the sequence of valencies $(\swarrow C)P(\nwarrow C)$ is satisfied the condition:
FA : P has no occurrences of $\swarrow C, \nwarrow C$ (i.e. $\swarrow C$ is the *first available* valency dual to $\nwarrow C$).

Example 2.



CDGs are more expressive than AB-grammars because they generate non-CF-languages (e.g. *MIX*, the language consisting of the strings over $\{a, b, c\}$ in which these symbols have the same number of occurrences). In this paper, the CDGs serve only as a background notion. Their strong definition as well as their mathematical properties may be found for instance in [8]. Here we only cite an equivalent definition of CDGs in terms of counter automata.

2.4. Abstract automata equivalent to CDGs

The automata equivalent to CDGs were defined by Karlov [12]. They have one stack and several completely independent counters (in fact, each pair of dual polarized CDG valencies corresponds to a unique 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

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

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. $v \in \mathbb{Z}^k$ (positive, null or negative integers) 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 $V \in \mathbb{N}^k$ (current counters' values are positive or null integers).

A computation step is the following transition relation:

$$\langle q, s, \gamma, V\rangle \ \vdash^i_A \ \langle q', s', \gamma', V'\rangle,$$

---

2 The order o is null on primitive types s.t. $o(X/Y) = o(Y\backslash X) = max(o(X), 1 + o(Y))$.

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

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

Example 3. The language $L = \{ w_1^n w_2^n w_3^n \mid n = 0, 1, \dots \}$ 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:

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

The equivalence of $RtPiCA^{(k)}$ and CDGs is proved in [12].

Theorem 7. *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 CDGs); 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. Classical 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 8 (*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 CDGs), are obtained as follows.

Definition 9. 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 $\mathbf{L^l}\ c^{P_1}\ [c \setminus \beta]^{P_2} \vdash [\beta]^{P_1P_2}$, 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 $\Omega^{\mathbf{l}}*\ [c^* \setminus \beta]^{P_2} \vdash [\beta]^{P_2}$, 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 $\mathbf{L^{l}}*\ c^{P_1}\ [c^* \setminus \beta]^{P_2} \vdash [c^* \setminus \beta]^{P_1P_2}$, 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 $\mathcal{T}_1, \mathcal{T}_2$;

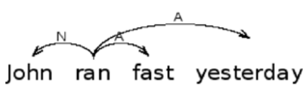
– we define similarly the function lfa_{iter} when the last rule is on the right, using $/$ and $\mathbf{L^r}$ instead of $\setminus$ and $\mathbf{L^l}$;

– (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 4. 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 :

$$\rho_1 : \dfrac{N \qquad \dfrac{\dfrac{\dfrac{\dfrac{[N \setminus S / A^*]\ \ A}{[N \setminus S / A^*]}\mathbf{I^r} \qquad A}{[N \setminus S / A^*]}\mathbf{I^r}}{[N \setminus S]}\Omega^{\mathbf{r}}*}{S}\mathbf{L^l}$$



(dependency structure)

3.3. Binary structures in $*$AB.

We introduce the definition of *-context, for a description of binary structures in $*$AB.

Definition 10. We say that B is an *-context of A, when $B = (G^*_{i,p'_i} \setminus \dots G^*_{i,1} \setminus A \,/\, D^*_{i,1} \dots / D^*_{i,p_i})$ where the sequences of iterated types (on the left, or on the right of A) are possibly empty. $B =_\star (A)_\star$ will mean that B is some *-context of A.

*-Context Rules. To simplify the presentation, we will also use elimination rules for *-contexts. The new rules will in a way incorporate the $\Omega^{\mathbf{r}}$ and $\Omega^{\mathbf{l}}$ rules.

The **-context elimination rules* are as follows :

${}_\star(X / Y)_\star\ ,\ {}_\star(Y)_\star \vdash X$	$({}_\star(\mathbf{L^r})_\star)$	${}_\star(Y)_\star\ ,\ {}_\star(Y \setminus X)_\star \vdash X$	$({}_\star(\mathbf{L^l})_\star)$
${}_\star(X / y^*)_\star\ ,\ {}_\star(y)_\star \vdash X / y^*$	$({}_\star(\mathbf{L^{r*}})_\star)$	${}_\star(y)_\star\ ,\ {}_\star(y^* \setminus X)_\star \vdash y^* \setminus X$	$({}_\star(\mathbf{L^{l*}})_\star)$
${}_\star(X / y^*)_\star \vdash X$	$({}_\star(\Omega^{\mathbf{r}})_\star)$	${}_\star(y^* \setminus X)_\star \vdash X$	$({}_\star(\Omega^{\mathbf{l}})_\star)$

System equivalence. Each rule above ${}_\star(R)_\star$ with antecedents ${}_\star(C_i)_\star$ is derivable from the original system, first applying several times $\Omega^{\mathbf{r}}$, $\Omega^{\mathbf{l}}$ according to the *-context and producing C_i, then applying rule R to C_i. Conversely, each elimination rule R is a case of ${}_\star(R)_\star$ with empty *-part in contexts. Therefore the two systems are equivalent.

Variant. The system consisting of ${}_\star(\mathbf{L^r})_\star, {}_\star(\mathbf{L^{r*}})_\star, {}_\star(\Omega^{\mathbf{r}})_\star, {}_\star(\mathbf{L^l})_\star, {}_\star(\mathbf{L^{l*}})_\star, {}_\star(\Omega^{\mathbf{l}})_\star$ is also equivalent to ${}_\star(\mathbf{L^r})_\star, {}_\star(\mathbf{L^{r*}})_\star, {}_\star(\mathbf{L^l})_\star, {}_\star(\mathbf{L^{l*}})_\star$, ${}_\star(\Omega)_\star$, where ${}_\star(\Omega)_\star$ is:

$$ {}_\star(X)_\star \vdash X \quad ({}_\star(\Omega)_\star) $$

this last version amounts to a simplification of ${}_\star(\Omega^{\mathbf{r}})_\star$ and ${}_\star(\Omega^{\mathbf{l}})_\star$.

Properties. if $\Delta, A, \Gamma \vdash X$ then $\Delta, {}_\star(A)_\star, \Gamma \vdash X$ as well (using $\Omega^{\mathbf{r}}*$ and $\Omega^{\mathbf{l}}*$).

Binary deduction trees. Each derivation tree in the original calculus can be transformed into a binary derivation tree involving only ${}_\star(\mathbf{L^r})_\star, {}_\star(\mathbf{L^{r*}})_\star, {}_\star(\mathbf{L^l})_\star, {}_\star(\mathbf{L^{l*}})_\star$, where the root is an *-context of S (written ${}_\star(S)_\star$).

We iteratively replace parts of the tree as follows:

- If the derivation tree has no binary rule, the succession of $\Omega^{\mathbf{r}}$ and $\Omega^{\mathbf{l}}$ is replaced by one application of ${}_\star(X)_\star \vdash X$ (where in fact $X = S$).
- If $\Omega^{\mathbf{r}}$ occurs before a binary rule R, we do the following replacements:

$$ \boxed{\dfrac{\dfrac{\vdots\delta_1 \atop (X_1 / Y_1) / y^*}{(X_1 / Y_1)}\,\Omega^{\mathbf{r}} \qquad {\vdots\delta_2 \atop Y_1}}{X_1}\,\mathbf{L^r}} \;\to\; \boxed{\dfrac{{\vdots\delta_1 \atop (X_1 / Y_1) / y^*} \quad {\vdots\delta_2 \atop Y_1}}{X_1}\,{}_\star(\mathbf{L^r})_\star} $$

$$ \boxed{\dfrac{\dfrac{\vdots\delta_1 \atop Z_1 / y^* =_\star ((X_1 / Y_1))_\star / y^*}{Z_1 =_\star ((X_1 / Y_1))_\star}\,\Omega^{\mathbf{r}} \qquad {\vdots\delta_2 \atop Z_2 =_\star (Y_1)_\star}}{X_1}\,{}_\star(\mathbf{L^r})_\star} \;\to\; \boxed{\dfrac{{\vdots\delta_1 \atop Z_1 / y^* = {}_\star((X_1 / Y_1))_\star / y^*} \quad {\vdots\delta_2 \atop Z_2 =_\star (Y_1)_\star}}{X_1}\,{}_\star(\mathbf{L^r})_\star} $$

  The transformations are similar for left elimination rules : $\mathbf{L^l}, {}_\star(\mathbf{L^{l*}})_\star$
- $\Omega^{\mathbf{l}}$ occurring before a binary rule is replaced similarly.

Remark. Using one of the system variants, we can eliminate the ${}_\star(\Omega^{\mathbf{r}})_\star$ and ${}_\star(\Omega^{\mathbf{l}})_\star$ in a way similar to $\Omega^{\mathbf{r}}$ and $\Omega^{\mathbf{l}}$.

4. A strict hierarchy

For each $k \in \mathbb{N}$, we are interested in classes $\mathcal{C}^k_{\langle constraint \rangle}$ of languages corresponding to k-valued grammars satisfying some $\langle constraint \rangle$. In this section we prove for some $\langle constraint \rangle$ (and for lexicons with at least 2 elements) that such families form strict hierarchies.

For instance, the first very easy observation considering the $*$AB calculus (denoted by * as class constraint) consists in that $\mathcal{C}^0_* \subsetneq \mathcal{C}^1_*$. Indeed, $\mathcal{C}^0_* = \emptyset$ and $\mathcal{C}^1_*$ contains the (finite) language $\{a\} = L_{*AB}(G)$ 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 [5] for the classes of k-valued classical categorial grammars. The proof scheme was as follows:

– Languages: for $k > 0$, $L_{AB,k} =_{def} \{a^iba^iba^i \mid 1 \leq i \leq 2k\}$.
– Grammars:[3] for $k > 0$,

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

– The language (for AB) of G_k is $L_{AB,k}$.
– 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 [13] 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 $k > 0$, $L_{ho,k} =_{def} \{abb\} \cup \{a^iba^iba^i \mid 1 \leq i \leq 2k\}$.
– Grammars: $k+1$-valued grammar $G'_k = \sigma(G_k)$ where G_k is as above, with substitution $\sigma = x := (S / y) / y$.
– The language (for NL) of G'_k is $L_{ho,k}$.
– Property: for $k > 0$, $L_{ho,k}$ 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 11. $\forall k \in \mathbb{N} \quad \mathcal{C}^k_{*,flat} \subsetneq \mathcal{C}^{k+1}_{*,flat}$.

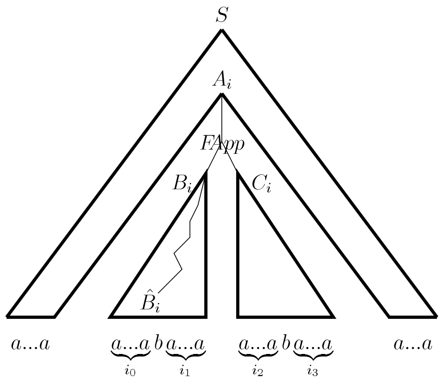
The detailed proof of this theorem needs some definitions and remarks.

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 previously described functor-argument structures.

Steps of proof.

1. Obviously, we have $\forall k \in \mathbb{N} \quad \mathcal{C}^k_* \subseteq \mathcal{C}^{k+1}_*$.
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 [5].
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 :
   $B_i =_\star (A_i / C^\delta_i)_\star$ (or $B_i =_\star (C^\delta_i \backslash A_i)_\star$) where δ is either $*$ or empty, and such that C'_i is a $*$context of C_i.

[3] In fact, the second type of a can be abbreviated as $S / x^iyx^iy^{i-1}$ and the second type of b can be abbreviated as $x^i \backslash (S / x^iyx^i)$.



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 centre 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 $w = \ldots ba^{j'}ba^j$ or $w = a^j ba^{j'} b \ldots$, 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 $B_i =_\star (A_i / C_i)_\star B_i = (G^*_{i,p'_i} \backslash \ldots G^*_{i,1} \backslash A_i / C_i / D^*_{i,1} \ldots / D^*_{i,p_i})$ (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 $X \rfloor Y$ as an abbreviation for $X \,/\, Y$ or for $Y \backslash X$ (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: $B_j = \ldots (B_i | D'_1 \ldots) | D'_n$ with $B_j =_\star (A_j / C_j)_\star$ (in the right case) ;
- - if B_i is a strict [4] 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[5] 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 $B_i \neq B_j$.

(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.

(e) 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 $k = 0$).

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.

---

[4] (Not equal to).
[5] Possibly equal to.

Theorem 12. $\forall k \in \mathbb{N} \quad \mathcal{C}_*^k \subsetneq \mathcal{C}_*^{k+1}$.

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

– Languages: for $k > 0$, $L_{ho,k} =_{def} \{abb\} \cup \{a^iba^iba^i \mid 1 \le i \le 2k\}$.
– Grammars: $k+1$-valued grammar $G'_k = \sigma(G_k)$ where G_k is as above, with substitution $\sigma = x := (S\ /\ y)\ /\ y$ (replacing x by the type). We can show $L_{*AB}(\sigma(G_k)) = L_{ho,k}$ as in [13] (see Annex).
– Property: for $k > 0$, $L_{ho,k}$ 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 $L_{ho,k}$ is not a k valued language, we proceed as in the previous section: we suppose the existence of a k-valued grammar G', with language $L_{ho,k}$ and we consider a deduction tree $\mathcal{T}_i$ for $a^iba^iba^i$ $(1 \le i \le 2k)$ and $\mathcal{T}_0$ for abb. For $0 \le i \le 2k$, 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: $\forall i \ne j : B_i \ne B_j$ (similarly to the previous subsection).
- $\forall i \ne j : \widehat{B_i} \ne \widehat{B_j}$ (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**. In this paper are studied two type calculi for categorial grammars using iterated types: one involving only flat types (i.e. the types of order 1) and the other using higher order types. We prove that for both the classes of k-valued categorial grammars induce strict hierarchies 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 in the way similar to that of the classical categorial grammars.

CDGs. In fact, our strict hierarchy theorem also extends to the CDGs with empty potentials, because every CDG with empty potentials may also be seen as a $*$AB grammar (of order 1). Therefore the hierarchy for CDGs with empty potentials does not collapse. The strict hierarchy problem for the unlimited CDGs is open.

Future work will concern iterated types extensions of other type logical grammars, e.g. the pregroup grammars.

Appendix. Semantic reasoning about language hierarchies

Useful models

Powerset residuated groupoids [14]. Let $(M, .)$ be a groupoid. Let $\mathcal{P}(M)$ denote the powerset of M. A *powerset residuated groupoid* over $(M, .)$ is the structure $(\mathcal{P}(M), \circ, \Rightarrow, \Leftarrow, \subseteq)$ such that for $X, Y \subseteq M$:

$$X \circ Y = \{x.y : x \in X, y \in Y\}$$
$$X \Rightarrow Y = \{y \in M : (\forall x \in X)\ x.y \in Y\}$$
$$Y \Leftarrow X = \{y \in M : (\forall x \in X)\ y.x \in Y\}.$$

Interpretation. Given a powerset residuated groupoid $(\mathcal{P}(M), \circ, \Rightarrow, \Leftarrow, \subseteq)$, an *interpretation* is a map from primitive types p to elements $[\![p]\!]$ in $\mathcal{P}(M)$ that is extended to types and sequences in the natural way :

$$[\![C_1 \setminus C_2]\!] = [\![C_1]\!] \Rightarrow [\![C_2]\!]\ ; \qquad [\![C_1\ /\ C_2]\!] = [\![C_1]\!] \Leftarrow [\![C_2]\!]\ ; \qquad [\![(C_1, C_2)]\!] = ([\![C_1]\!] \circ [\![C_2]\!]).$$

By a model property for NL : If $\Gamma \vdash_{NL} C$ then $[\![\Gamma]\!] \subseteq [\![C]\!]$.

Description of $L(\sigma(G_k))$ using models, (following [13])

For the language description ($L_*(\sigma(G_k)) = L_{AB}(\sigma(G_k))$, case order>1), we consider the $k+1$-valued grammar $\sigma(G_k)$ where G_k is as above, with substitution $\sigma = x := (S\ /\ y)\ /\ y$, and we show $L_{NL}(\sigma(G_k)) = L_{ho,k}$.

- We show that $L_{ho,k} \subseteq L(_{NL}(\sigma(G_k)))$ by:
  For $(i = 0, abb)$: $((((S \,/\, y) \,/\, y), y), y) \vdash S$ we write $F_0 = ((S \,/\, y) \,/\, y)$.
  For $(i \leq k, a^iba^iba^i)$:$(\ldots(S \,/\, x^iyx^iyx^{i-1}, \underbrace{x)\ldots, x)}_{i-1}, y), \underbrace{x)\ldots, x)}_{i}, y), \underbrace{x)\ldots, x)}_{i} \vdash S$ and let $F_i = S \,/\, x^iyx^iyx^{i-1}$ denote the corresponding type of a.
  For $(i > k, a^iba^iba^i)$: $\underbrace{(x, \ldots, (x}_{i}, x^i \setminus S \,/\, x^iyx^i, \underbrace{x)\ldots, x)}_{i}, y, \underbrace{x)\ldots, x)}_{i} \vdash S$
  and let $F_i = x^i \setminus S \,/\, x^iyx^i$ denote the corresponding type of b.
- To show that $L_{NL}(\sigma(G_k)) \subseteq L_{NL,k}$ we consider the following powerset residuated groupoid on V^* (also with unit):
  $[\![S]\!] = L_{ho,k}$, $[\![y]\!] = \{b\}$;
  we then calculate the type images of $\sigma(F_i)$ (see above) :
  $[\![\sigma(F_0)]\!] = \{a\}$ (with $[\![(S \,/\, y)]\!] = \{ab\}$)
  if $(i \leq k)$ then $[\![\sigma(F_i)]\!] = \{a\}$,
  if $(i' > k)$ then $[\![\sigma(F_{i'})]\!] = \{b\}$
  hence the language inclusion ($\Gamma \vdash S$ implies $[\![\Gamma]\!] \subseteq [\![S]\!] = L_{ho,k}$).
- $L_{AB}(\sigma(G_k)) = L_{NL}(\sigma(G_k))$ is already established in [13]. This can obtained from (a) $L_{ho,k} \subseteq L_{AB}(\sigma(G_k)) \subseteq L_{NL}(\sigma(G_k))$ (same parses as above, and AB proofs hold in NL) and (b) $L_{ho,k} = L_{NL}(\sigma(G_k))$ as recalled above.

References

[1] Y. Bar-Hillel, A quasi arithmetical notation for syntactic description, Language 29 (1953) 47–58.
[2] J. Lambek, The mathematics of sentence structure, American mathematical monthly 65.
[3] A.K. Joshi, Y. Shabes, Tree-adjoining grammars and lexicalized grammars, in: Tree Automata and LGS, Elsevier Science, Amsterdam, 1992.
[4] E. Gold, Language identification in the limit, Information and control 10 (1967) 447–474.
[5] M. Kanazawa, 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.
[6] A. Dikovsky, Dependencies as categories, in: Recent Advances in Dependency Grammars, COLING'04 Workshop, 2004, pp. 90–97.
[7] D. Béchet, A. Dikovsky, A. Foret, E. Moreau, On learning discontinuous dependencies from positive data, in: P. Monachesi (Ed.), Proc. of the 9th Intern. Conf. Formal Grammar 2004, FG 2004, 2004, pp. 1–16.
[8] M. Dekhtyar, A. Dikovsky, Generalized categorial dependency grammars, in: Trakhtenbrot/Festschrift, in: LNCS, vol. 4800, Springer, 2008, pp. 230–255.
[9] I. Mel'čuk, Dependency Syntax, SUNY Press, Albany, NY, 1988.
[10] D. Béchet, A. Dikovsky, A. Foret, E. Garel, Introduction of option and iteration into pregroup grammars, in: C. Casadio, J. Lambek (Eds.), Computational Algebric Approaches to Morphology and Syntax, Polimetrica, Monza (Milan), Italy, 2008, pp. 85–107.
[11] D. Béchet, A. Dikovsky, A. Foret, Two models of learning iterated dependencies, in: M. Egg, P. de Groote, L. Kallmeyer, M.-J. Nederhof (Eds.), Proceedings of the 15th International Conference on Formal Grammar (FG10), Copenhagen, Denmark, 2010, pp. 1–16.
[12] B. Karlov, Abstract automata and a normal form for categorial dependency grammars, in: Proceedings of LACL 2012, in: LNCS, vol. 7351, Springer, Nantes, France, 2012.
[13] D. Bechet, A. Foret, k-valued non-associative Lambek grammars (without product) form a strict hierarchy of languages, in: Proceedings of LACL 2005, in: LNCS (LNAI), vol. 3492, Springer, 2005, pp. 1–17.
[14] W. Buszkowski, Mathematical linguistics and proof theory, in: J. van Benthem, A. ter Meulen (Eds.), Handbook of Logic and Language, North-Holland, Elsevier, Amsterdam, 1997, pp. 683–736. Ch. 12.