# k-Valued Non-Associative Lambek Grammars (without Product) Form a Strict Hierarchy of Languages

Denis Béchet[1] and Annie Foret[2]

[1] LINA – FRE 2729
Université de Nantes & CNRS
2, rue de la Houssiniére — BP 92208
44322 Nantes Cedex 03, France
Denis.Bechet@univ-nantes.fr
[2] IRISA – Université de Rennes 1
Campus Universitaire de Beaulieu
Avenue du Général Leclerc
35042 Rennes Cedex, 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. This paper establishes the relevance of this notion for two related grammatical systems. In the first part, the classes of $k$-valued non-associative Lambek ($NL$) grammars without product is proved to define a strict hierarchy of languages. The second part introduces the notion of generalized functor argument for non-associative Lambek ($NL_\emptyset$) calculus without product but allowing empty antecedent and establishes also that the classes of $k$-valued $NL_\emptyset$ grammars without product form a strict hierarchy of languages.

## 1 Introduction

The field of natural language processing includes lexicalized grammars such as classical categorial 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 a particular model of lexicalized grammars and their corresponding languages, this definition forms a (strict) hierarchy of classes of grammars when $k$ increases. This hierarchy of grammars corresponds to 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 satify some property when the whole class does not satify it. In particular, this notion is

DRAFT

important for a lot of learnability results in Gold's model [4]. Usually, to find a positive result, some kind of restriction is necessary because very often the whole class of grammars corresponds to (at least) context free languages and this class is known to be unlearnable no matter which grammatical system is used[3].

Usually, when the learnability can be established for a particular $k$-valued class of grammars, this property can also be proved for any $k \in \mathbb{N}$. When this assumption is true, for each $k \in \mathbb{N}$, there exists a learning algorithm that learns the grammars of this class from positive examples even if the whole class is not learnable (there does not exist a universal learning algorithm for all $k \in \mathbb{N}$): we hope that all the interesting grammars (the grammars for natural languages, for instance) are in a particular $k$-valued class.

In this context, we can wonder if, for a particular system of lexicalized grammars and languages, the class of classes of languages forms a strict hierarchy. If this assumption is not verified for a system, it usually means that for a certain $k$ the class of $k$-valued languages is the same as the whole class: the hierarchy is truncated. The other possibility which is very rare consists in a hierarchy that has infinite steps, some steps corresponding to several contiguous integers. In this context, the proof that the hierarchy is not strict is usually a bad news for the concept of $k$-valued grammars corresponding to a system. For instance, the classes of $k$-valued classical categorial grammar form a strict hierarchy of languages [5] and for each $k \in \mathbb{N}$, the class of $k$-valued classical categorial grammar is learnable from positive examples.

In the paper, we are interested to prove that non-associative Lambek grammars without product (written *NL*) and non-associative Lambek grammars without product but with empty antecedent (written $NL_\emptyset$) form two strict hierarchies of classes of languages. The results give a direct justification of the notion of $k$-valued grammars for both systems. The paper also recalls a useful alternative presentation of *NL* using generalized AB deductions (written *GAB*) and generalized functor-argument structures (written *FA*) that were used in [6] for defining a learning algorithm but are a central notion in the paper for proving the strict hierarchy. For $NL_\emptyset$, the paper introduces similar notions: generalized AB deductions with empty applications (written $GAB_\emptyset$) and generalized functor-argument structures with empty arguments (written $FA_\emptyset$). This presentation is also proved, in the paper, to be equivalent to $NL_\emptyset$ and the hierarchy theorem is given.

The paper is organized as follows. Section 2 gives some background knowledge on non-associative Lambek categorial grammars (without product). Section 3 focuses on (recently defined) structures, with alternative deduction rules for NL-grammars (without product) as generalized FA-structures; in fact these rules are extensions of the cancellation rules of classical categorial grammars that lead to the generalization of FA-structures used here. Section 4 presents the proof that the class of $k$-valued non-associative Lambek categorial grammars without

[3] The class of context free language has a limit point: $\exists L, (L_i)_{i\in\mathbb{N}}$ such that $L = \bigcup_{i\in\mathbb{N}} L_i$ and $\forall i \in \mathbb{N}\ L_i \subsetneq L_{i+1}$. This property entails that any class of grammars that corresponds to (a superclass of) context free languages (of strings) is not learnable from positive examples

product form a strict hierarchy. Section 5 considers the $NL_\emptyset$ variant of NL ; we define generalized functor-argument structures for this variant and a strict hierarchy theorem; we also revisit the case of AB (classical categorial grammars). Section 6 concludes.

## 2 Background

### 2.1 Categorial Grammars

The reader not familiar with Lambek Calculus and its non-associative version will find nice presentation in the first articles written by Lambek [2, 7] or more recently in [8–13]. We use in the paper two variants of Lambek calculus: non-associative Lambek calculus without product with ($NL_\emptyset$) and without ($NL$) empty antecedent.

**Definition 1 (Types).** *The* types *$Tp$, or formulas, are generated from a set of* primitive types *$Pr$, or atomic formulas, by two binary connectives*[4] *"/" (over) and "\" (under):*

$$Tp ::= Pr \mid Tp\backslash Tp \mid Tp/Tp$$

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

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

*If $X \in I(a)$, 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 $a \in \Sigma$, $I(a)$ has at most $k$ elements. A* rigid categorial grammar *is a 1-valued categorial grammar.*

### 2.2 Non-associative Lambek Calculi *NL* and $NL_\emptyset$

***NL*/$NL_\emptyset$ derivation $\vdash_{NL}/\vdash_{NL_\emptyset}$** As a logical system, we use Gentzen-style sequent presentation. A sequent $\Gamma \vdash A$ is composed of a binary tree of formulas $\Gamma$ (the set of such trees is noted $\mathcal{T}_{Tp}$) which is the antecedent configuration and a succedent formula $A$. A context $\Gamma[\cdot]$ is a binary tree of formulas with a hole. For $X$, a formula or a binary tree of formulas, $\Gamma[X]$ is the binary tree obtained from $\Gamma[\cdot]$ by filling the hole with $X$. $\Gamma$ can be empty in $NL_\emptyset$: $\Gamma$ belongs to $\mathcal{T}_{Tp} \cup \{\emptyset\}$. In fact, to completely define the rules of $NL_\emptyset$, we use two equivalence relations on binary trees of formulas and the empty set[5]:

$$\Gamma[(\emptyset, \Delta)] \equiv_{NL_\emptyset} \Gamma[\Delta] \equiv_{NL_\emptyset} \Gamma[(\Delta, \emptyset)]$$

[4] no product connective is used in the paper

[5] One can define $NL_\emptyset$ without the two equivalence relations by specializing the rules when one of the antecedents is empty but this gives a much longer definition

Draft

**Definition 3** ($NL/NL_\emptyset$). *A sequent is valid in $NL/NL_\emptyset$ and is noted $\Gamma \vdash_{NL} A/\Gamma \vdash_{NL_\emptyset} A$ iff $\Gamma \vdash A$ can be deduced from the following rules:*

$$\frac{}{A \vdash A}\,\mathbf{Ax} \qquad \frac{(\Gamma, B) \vdash A}{\Gamma \vdash A/B}\,/R \qquad \frac{(A, \Gamma) \vdash B}{\Gamma \vdash A\backslash B}\,\backslash R$$

$$\frac{\Gamma \vdash A \quad \Delta[A] \vdash B}{\Delta[\Gamma] \vdash B}\,\mathbf{Cut} \qquad \frac{\Gamma \vdash A \quad \Delta[B] \vdash C}{\Delta[(B/A, \Gamma)] \vdash C}\,/L \qquad \frac{\Gamma \vdash A \quad \Delta[B] \vdash C}{\Delta[(\Gamma, A\backslash B)] \vdash C}\,\backslash L$$

**Cut elimination.** We recall that the cut rule can be eliminated in $\vdash_{NL}$ and in $\vdash_{NL_\emptyset}$ : every derivable sequent has a cut-free derivation.

***NL/NL$_\emptyset$* languages.** $\mathcal{E}^+$ denotes the set of non-empty strings over $\mathcal{E}$. $\mathcal{T}_\mathcal{E}$ is the set of (non-empty) well-bracketed lists (binary trees) of elements of $\mathcal{E}$.

**Definition 4 (Yield).** *If $T$ is a tree where the leaves are elements of a set $\mathcal{E}$, $yield_\mathcal{E}(T) \in \mathcal{E}^+$ is the list of leaves of $T$.*

This notation will be used for well-bracketed lists of words $yield_\Sigma$, for binary trees of formulas $yield_{Tp}$ and will be extended to *FA* structures (see further Definition 7).

**Definition 5 (Language).** *Let $G = (\Sigma, I, S)$ be a categorial grammar.*

- *$G$* generates a well-bracketed list of words *$T \in \mathcal{T}_\Sigma$ (in $NL/NL_\emptyset$ model) iff there exists $\Gamma$ a binary tree of types, $c_1, \ldots, c_n \in \Sigma$ and $A_1, \ldots, A_n \in Tp$ such that:*
$$\begin{cases} G : c_i \mapsto A_i \ (1 \leq i \leq n) \\ \Gamma = T[c_1 \to A_1, \ldots, c_n \to A_n] \\ \Gamma \vdash_{NL} S/\Gamma \vdash_{NL_\emptyset} S \end{cases}$$
*where $T[c_1 \to A_1, \ldots, c_n \to A_n]$ means the binary tree obtained from $T$ by substituting the left to right occurrences of $c_1, \ldots, c_n$ by $A_1, \ldots, A_n$.*
- *$G$* generates a string *$c_1 \cdots c_n \in \Sigma^+$ iff there exists $T \in \mathcal{T}_\Sigma$ such that $yield_\Sigma(T) = c_1 \cdots c_n$ and $G$ generates $T$.*
- *The* language of well-bracketed lists of words *of $G$, written $\mathcal{BL}_{NL}(G)/\mathcal{BL}_{NL_\emptyset}(G)$, is the set of well-bracketed lists of words generated by $G$.*
- *The* language of strings *corresponding to $G$, written $\mathcal{L}_{NL}(G)/\mathcal{L}_{NL_\emptyset}(G)$, is the set of strings generated by $G$.*

One interest of *NL* when compared to classical categorial grammars lies in its possibility to easily encode a restriction on the use of a basic category. For instance when we want to distinguish between a noun phrase and pronouns in subject position or object position, we can proceed as follows.

*Example 1.* Let $\Sigma_1 = \{John, Mary, likes, he, she, him, her\}$ and let $Pr_1 = \{S, N, X_1, X_2\}$. We define the following rigid grammar:

$$G_1 = \begin{cases} John, Mary \mapsto N & ; he, she \mapsto N_1; \\ him, her \quad\ \mapsto N_2 & ; likes \quad \mapsto N_1\backslash(S/N_2). \end{cases}$$

where $N_1 = X_1/(N\backslash X_1)$ and $N_2 = X_2/(N\backslash X_2)$.

We get: $((He\ likes)\ Mary) \in \mathcal{BL}_{NL}(G_1)$ but: $John\ likes\ she \notin \mathcal{L}_{NL}(G_1)$

Draft

With $NL_\emptyset$, we can introduce a restrictive form of optional arguments; "little", in the following example, is optionally associated to proper nouns. This is not possible directly with $NL$.

*Example 2.* Let $\Sigma_2 = \{John, Mary, likes, little\}$ and let $Pr_2 = \{S, N, L\}$.
We define: $G_2 = \begin{cases} John \mapsto (L\backslash L)\backslash N \; ; \; Mary \mapsto (L\backslash L)\backslash N \\ little \mapsto L\backslash L \qquad ; \; likes \; \mapsto N\backslash(S/N) \end{cases}$
$G_2$ is a rigid (or 1-valued) grammar. We can prove that $(((L\backslash L)\backslash N,\ N\backslash(S/N)),\ (L\backslash L)\backslash N) \vdash_{NL_\emptyset} S$ and $(((L\backslash L)\backslash N,\ N\backslash(S/N)),\ (L\backslash L, (L\backslash L)\backslash N)) \vdash_{NL_\emptyset} S$.
Thus, we get:

$$John\ likes\ Mary \quad \in \mathcal{L}_{NL_\emptyset}(G_2) \quad ; \quad John\ likes\ little\ Mary \quad \in \mathcal{L}_{NL_\emptyset}(G_2)$$
$$((John\ likes)\ Mary) \in \mathcal{BL}_{NL_\emptyset}(G_2)\ ;\ ((John\ likes)\ (little\ Mary)) \in \mathcal{BL}_{NL_\emptyset}(G_2)$$

### 2.3 Some useful Models.

**Powerset residuated groupoids and semi-groups.** 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 $X, Y \subseteq M$:

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

If $(M, .)$ has a unit $I$ ( that is : $\forall x \in M : I.x = x.I = x$), then the above structure is a *powerset residuated groupoid with unit* (it has $\{I\}$ as unit).
**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 :
$[\![C_1\backslash C_2]\!] = [\![C_1]\!] \Rightarrow [\![C_2]\!]$ ; $[\![C_1/C_2]\!] = [\![C_1]\!] \Leftarrow [\![C_2]\!]$ ; $[\![(C_1, C_2)]\!] = ([\![C_1]\!] \circ [\![C_2]\!])$
By a model property for $NL$ : If $\Gamma \vdash_{NL} C$ then $[\![\Gamma]\!] \subseteq [\![C]\!]$
If $(M, .)$ is a groupoid with a unit $I$, we add $[\![\emptyset]\!] = \{I\}$ for the empty sequence $\emptyset$ and get a model property for $NL_\emptyset$ : if $\Gamma \vdash_{NL_\emptyset} C$ then $[\![\Gamma]\!] \subseteq [\![C]\!]$

## 3 GAB deductions and generalized FA-structures

In this section we focus on (recently defined) structures [6], with alternative deduction rules for NL-grammars (without product) as generalized FA-structures; 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 $\Sigma$ or a set of types such as $Tp$.

Draft

**Definition 6 (*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 FApp (forward application) or BApp (backward application):*

$$\mathcal{FA}_{\mathcal{E}} ::= \mathcal{E} \mid FApp(\mathcal{FA}_{\mathcal{E}}, \mathcal{FA}_{\mathcal{E}}) \mid BApp(\mathcal{FA}_{\mathcal{E}}, \mathcal{FA}_{\mathcal{E}})$$

**Definition 7 (Tree yield).** *The well-bracketed list of words obtained from a FA structure $F$ over $\mathcal{E}$ by forgetting FApp and BApp labels is called the* tree yield *of $F$ over $\mathcal{E}$ (notation $tree_{\mathcal{E}}(F)$).*

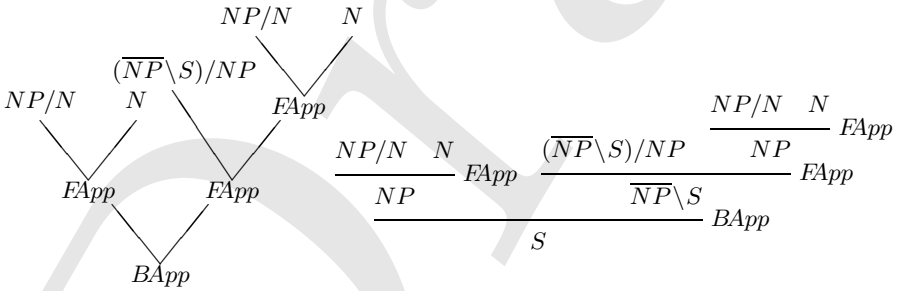
### 3.2 *GAB* deductions

**Definition 8 (*GAB* Deduction).** Generalized *AB* deductions *(GAB deductions) over Tp are the deductions built from formulas on Tp (the base case) using the following conditional rules ($C \vdash_{NL} B$ must be valid in NL):*

$$\frac{A/B \quad C}{A}\ FApp \qquad \frac{C \quad B\backslash A}{A}\ BApp$$
$$C \vdash_{NL} B \text{ valid in NL}$$

*GAB* deductions can be seen as a generalization of *AB* deductions in the following sense: for *AB* application rules $C$ and $B$ must be the same formula.

**Definition 9 (*FA* structure of a *GAB* deduction).** *To each GAB deduction $\mathcal{P}$, we associate a FA structure, written $FA_{Tp}(\mathcal{P})$, such that each internal node corresponds to the application of a rule in $\mathcal{P}$ and is labelled by the name of this rule and where the leaves are the same as in $\mathcal{P}$.*




Here, $\overline{NP} = X/(NP\backslash X)$ and thus $NP \vdash_{NL} \overline{NP}$

**Definition 10 (*GAB* Deductions of $F \vdash_{GAB} A$ or $\Gamma \vdash_{GAB} A$).**

- *For a FA structure $F \in \mathcal{FA}_{Tp}$ (over types) and $A \in Tp$, we say that $\mathcal{P}$ is a GAB deduction*[6] *of $F \vdash_{GAB} A$ when $A$ is the type of the conclusion of $\mathcal{P}$ and when $FA_{Tp}(\mathcal{P}) = F$.*
- *For a tree $\Gamma \in \mathcal{T}_{Tp}$ (over types) and $A \in Tp$, we say that $\mathcal{P}$ is a GAB deduction of $\Gamma \vdash_{GAB} A$ when $A$ is the type of the conclusion of $\mathcal{P}$ and when $tree_{Tp}(FA_{Tp}(\mathcal{P})) = \Gamma$.*

[6] in fact, given a *FA* structure $F$, there is at most one *GAB* deduction $\mathcal{P}$ s.t. $FA_{Tp}(\mathcal{P}) = F$ ($FA_{Tp}$ is injective)

Draft

***GAB* Languages** Similarly to classical categorial grammars, we can associate to each categorial grammar a language of *FA* structures.

**Definition 11 (*GAB* Languages).** *Let $G = (\Sigma, I, S)$ be a categorial grammar over Tp :*

- *$G = (\Sigma, I, S)$* generates a *FA* structure *$F \in \mathcal{FA}_\Sigma$ (in the GAB derivation model) iff there exists a GAB derivation of a FA structure $D \in \mathcal{FA}_{Tp}$, $c_1, \ldots, c_n \in \Sigma$ and $A_1, \ldots, A_n \in Tp$ such that:*

$$\begin{cases} G : c_i \mapsto A_i \ (1 \leq i \leq n) \\ D = F[c_1 \to A_1, \ldots, c_n \to A_n] \\ D \vdash_{GAB} S \end{cases}$$

  *where $F[c_1 \to A_1, \ldots, c_n \to A_n]$ means the FA structure obtained from $F$ by substituting respectively the left to right occurrences of $c_1, \ldots, c_n$ by $A_1, \ldots, A_n$.*
- *$G$* generates a well-bracketed list of words *$T \in \mathcal{T}_\Sigma$ iff there exists $F \in \mathcal{FA}_\Sigma$ such that $tree_\Sigma(F) = T$ and $G$ generates $F$.*
- *$G$* generates a string *$c_1 \cdots c_n \in \Sigma^+$ iff there exists $F \in \mathcal{FA}_\Sigma$ such that $yield_\Sigma(tree_\Sigma(F)) = c_1 \cdots c_n$ and $G$ generates $F$.*
- *The* language of *FA* structures *corresponding to $G$, written $\mathcal{FL}_{GAB}(G)$, is the set of FA structures generated by $G$.*
- *The* language of well-bracketed lists of words *corresponding to $G$, written $\mathcal{BL}_{GAB}(G)$, is the set of well-bracketed lists of words generated by $G$.*
- *The* language of strings *corresponding to $G$, written $\mathcal{L}_{GAB}(G)$, is the set of strings generated by $G$.*

*Example 3.* If we take the categorial grammar that is defined in Example 1, we get:

$$\begin{aligned} He\ likes\ Mary &\in \mathcal{L}_{GAB}(G_1) \\ ((He\ likes)\ Mary) &\in \mathcal{BL}_{GAB}(G_1) \\ FApp(BApp(He, likes), Mary) &\in \mathcal{FL}_{GAB}(G_1) \end{aligned}$$

because we can build the following *GAB* deduction (where $N_2 = X_2/(N \setminus X_2)$ that entails $N \vdash N_2$):

$$\dfrac{\dfrac{\overbrace{N_1}^{He} \quad \overbrace{N_1 \setminus (S/N_2)}^{likes}}{S/N_2}\, BApp \quad \overbrace{N}^{Mary}}{S}\, FApp$$

however: $Mary\ likes\ he \notin \mathcal{L}_{GAB}(G_1)$

### 3.3 *NL* and *GAB* languages

In fact, there is a strong correspondence between *GAB* deductions and *NL* derivations. In particular Theorem 1 shows that the respective string languages and binary tree languages are the same.

**Theorem 1 ([6]).** *If $A$ is an atomic formula, $\Gamma \vdash_{GAB} A$ iff $\Gamma \vdash_{NL} A$*

**Corollary 1.** *$\mathcal{BL}_{NL}(G) = \mathcal{BL}_{GAB}(G)$ and $\mathcal{L}_{NL}(G) = \mathcal{L}_{GAB}(G)$*

Draft

## 4 A strict hierarchy

For each $k \in \mathbb{N}$, we can consider the class $\mathcal{C}_{NL}^k$ of languages corresponding to $k$-valued non-associative Lambek grammars (without product) that is the grammars with at most $k$ types associated to each word of the lexicon. This section proves that this family forms a strict hierarchy (if the lexicon has at least 2 elements):

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

For instance, a first very easy result is given by the fact that $\mathcal{C}_{NL}^0 \subsetneq \mathcal{C}_{NL}^1$ because $\mathcal{C}_{NL}^0 = \emptyset$ and $\mathcal{C}_{NL}^1$ contains the (finite) language $\{a\} = \mathcal{L}_{NL}(G)$ for the rigid grammar $G : a \mapsto S$.

### 4.1 Overview

**Previous works.** 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 $k > 0$, $L_{AB,k} =_{def} \{a^i b a^i b a^i \mid 1 \leq i \leq 2k\}$
- Grammars:[7] for $k > 0$,

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

- 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.

**Towards NL.** We can show easily that the languages of grammars $G_k$ is the same when we consider the NL calculus instead of the AB rules : because the order of the types in the grammars are at most one, and in such a case it is well-known that the NL-languages and the AB-languages associated to a grammar are the same.

This shows that the languages $L_{AB,k}$ are also $(k+1)$-valued languages for NL. It is thus natural to ask whether they are $k^-$valued for NL as well. In fact, when we proceed to a proof based on functor-argument structures, some of the arguments are no longer valid for NL, in the case of GAB-deductions.

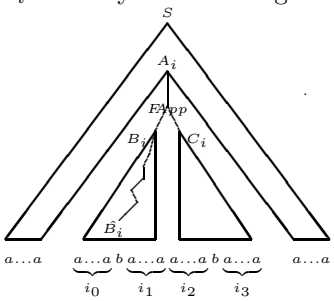
We have thus constructed another language, based on the previous one, with some very similar proof steps. One key difference appears at stage 4.(e)(f) that does not apply to $L_{AB,k}$ that has only $2k$ words. An important point of our treatment for *NL* is precisely that the language constructed is both $k+1$-valued and with $2k+1$ words.

The proof scheme is as follows:

[7] 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 $x^i\backslash(S/x^i y x^i)$

Draft

1. Obviously, we have $\forall k \in \mathbb{N} \quad \mathcal{C}_{NL}^k \subseteq \mathcal{C}_{NL}^{k+1}$
2. For $k > 0$, we consider $L_{NL,k} =_{def} \{abb\} \cup \{a^i ba^i ba^i \mid 1 \leq i \leq 2k\}$
3. We prove that $L_{NL,k}$ is a $(k+1)$-valued language for NL languages.
   Proof: for $k > 0$, We define a $k+1$-valued grammar and show that it is associated to $L_{NL,k}$. This part of the proof uses models of $NL$ (see 4.2 below).
4. We prove that $L_{NL,k}$ is not a $k$-valued language for NL languages.
   Proof : suppose $G$ is a $k$-valued grammar with language $L_{NL,k}$
   (a) For each element of $L_{NL,k}$, there exists a GAB deduction: $\mathcal{T}_i$ for $a^i ba^i ba^i$ $(1 \leq i \leq 2k)$ and $\mathcal{T}_0$ for $abb$.
   (b) 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 $b$:




   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 $B_i = A_i/D_i$ (or $B_i = D_i \setminus A_i$) and $C_i$ with $C_i \vdash_{NL} D_i$ the antecedents of $A_i$ in $\mathcal{T}_i$ and we define $\widehat{B_i}$ as the type in $G$ "providing" $B_i$ (following functors) in $\mathcal{T}_i$.
   (c) We prove that: $\forall i, j : B_i \neq B_j$ (see 4.2 below)
   (d) and: $\forall i, j : \widehat{B_i} \neq \widehat{B_j}$ (see 4.2 below)
   (e) As a consequence, we need $2k+1$ distinct $\widehat{B_i}$.
   (f) Contradiction: $2k+1$ distinct $\widehat{B_i}$ are needed with a $k$-valued grammar with an appropriate lexicon of 2 words ($a$ and $b$).
5. Thus $\forall k > 0 \quad \mathcal{C}_{NL}^k \neq \mathcal{C}_{NL}^{k+1}$ (we have also seen in the introduction to the section that the property is also true for $k = 0$).

### 4.2 Details of proof

– For (4.c): let $y_{ce}(X_i)$ denote the center part of the yield with root $X_i$ (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 $\forall i : y_{ce}(B_i) + y_{ce}(C_i) = i$.
Suppose (by contradiction) (i) $B_i = B_j$, for some $i \neq j$ ; from (i) we get $D_i = D_j$, and $C_i \vdash D_i$ then also $C_i \vdash D_j$, $C_j \vdash D_i$.
Since $i \neq j$, either $y_{ce}(B_i) \neq y_{ce}(B_j)$ or $y_{ce}(C_i) \neq y_{ce}(C_j)$.
Suppose first (ii) $y_{ce}(B_i) \neq y_{ce}(B_j)$ ; from (ii) replacing in $T_j$, $(j \neq 0)$, $B_j$ by $B_i$ is a parse of a word $...ba^{j'} ba^j$ or $a^j ba^{j'} b...$, where $j' = y_{ce}(B_i) + y_{ce}(C_j)$ this word is not in $L_{NL,k}$ since $j' = y_{ce}(B_i) + y_{ce}(C_j) \neq j$ this contradicts the assumption that $G$ has $L_{NL,k}$ as language (for $NL$).
Suppose instead (ii)' $y_{ce}(C_i) \neq y_{ce}(C_j)$, replacing in $T_j$, $(j \neq 0)$, $C_j$ by $C_i$ yields a similar word not in $L_{NL,k}$ with $j' = y_{ce}(B_j) + y_{ce}(C_i)$ occurrence of $a$ between the b and $j' \neq j$, (ii)' also brings a contradiction.
Therefore (i) is not possible.

– For (4.d): we use the notation $X|Y$ as an abbreviation for $X/Y$ or for $Y\backslash X$ (functor first). We show $\forall i,j : \widehat{B_i} \neq \widehat{B_j}$.
Suppose $\widehat{B_i} = \widehat{B_j}$, one (say $B_i$) is a head subtype of the other ($B_j$) then of the form: $B_j = ...(B_i|D'_1...)|D'_n = A_j|D_j$
we then get $A_j$ in the subtree ending in $B_i$ in $T_i$, which is impossible since the yield would have three $b$ instead of two.
– For (3) the language description, 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 $\mathcal{L}_{NL}(\sigma(G_k)) = L_{NL,k}$.
  - We show that $L_{NL,k} \subseteq \mathcal{L}(_{NL}(\sigma(G_k)))$ by:
    For $(i = 0, abb) : ((((S/y)/y), y), y) \vdash S$ we write $F_0 = ((S/y)/y)$.
    For $(i \leq k, a^iba^iba^i)$:$(..(S/x^iyx^iyx^{i-1}, \underbrace{x)..,x)}_{i-1}, y), \underbrace{x)...,x)}_{i}, y), \underbrace{x)...,x)}_{i} \vdash S$
    and let $F_i = S/x^iyx^iyx^{i-1}$ denote the corresponding type of $a$.
    For $(i > k, a^iba^iba^i) : \underbrace{(x,..,(x}_{i}, x^i\backslash S/x^iyx^i, \underbrace{x)...,x)}_{i}, y, \underbrace{x)...,x)}_{i} \vdash S$
    and let $F_i = x^i\backslash S/x^iyx^i$ denote the corresponding type of $b$.
  - To show that $\mathcal{L}_{NL}(\sigma(G_k)) \subseteq L_{NL,k}$ we consider the following powerset residuated groupoid on $V^*$ (also with unit):
    $[\![S]\!] = L_{NL,k}$, $[\![y]\!] = \{b\}$ ;
    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 $[\![\Gamma]\!] \subseteq [\![S]\!] = L_{NL,k}$).

# 5 Variants

## 5.1 A variant of NL

In this section, we propose an adaptation of *GAB* deductions to the variant $NL_\emptyset$ (without product). The presentation of $NL_\emptyset$ using $GAB_\emptyset$ and generalized $FA_\emptyset$ structures is original. This is done by substituting *NL* by $NL_\emptyset$ for the conditions on the premises of the *GAB* rules and also by adding two new rules written $FApp_\emptyset$ and $BApp_\emptyset$ that correspond to a "pseudo-application" of a functor to an "empty" argument. In the second part of the section, we show how to adapt the previous hierarchy construction to $NL_\emptyset$ with the use of $GAB_\emptyset$ deductions.

**Defining $GAB_\emptyset$ deductions for $NL_\emptyset$** Like *NL*, $NL_\emptyset$ can be presented as a $GAB_\emptyset$ deduction system. This construction enables us to define generalized functor-argument structures for $NL_\emptyset$ proofs and as elements of languages of structures. The FA structures have two new unary combinators $FApp_\emptyset$ and $BApp_\emptyset$ that correspond to the application of an empty argument.

Draft

**Definition 12 ($GAB_\emptyset$ Deduction for $NL_\emptyset$).** Generalized *AB* deductions with empty applications *for $NL_\emptyset$ ($GAB_\emptyset$ deductions) over Tp are the deductions built from formulas on Tp (the base case) using the following conditional rules ($C \vdash_{NL_\emptyset} B$ must be valid in $NL_\emptyset$ in the first two rules and $\vdash_{NL_\emptyset} B$ must be valid in $NL_\emptyset$ in the last two rules):*

$$\frac{A/B \quad C}{A}\ FApp \qquad \frac{C \quad B\backslash A}{A}\ BApp$$
$$C \vdash_{NL_\emptyset} B \text{ valid in } NL_\emptyset$$

$$\frac{A/B}{A}\ FApp_\emptyset \qquad \frac{B\backslash A}{A}\ BApp_\emptyset$$
$$\vdash_{NL_\emptyset} B \text{ valid in } NL_\emptyset$$

We now extend the notion of *FA* structure, written $FA_\emptyset$ so as to include the case of an empty antecedent (empty argument).

**Definition 13 ($FA_\emptyset$ structures ).** *Let $\mathcal{E}$ be a set, a $FA_\emptyset$* structure *over $\mathcal{E}$ is a tree where each leaf is labelled by an element of $\mathcal{E}$ and each internal node is labelled by FApp (forward application), BApp (backward application), $FApp_\emptyset$ (forward empty application), $BApp_\emptyset$ (backward empty application):*

$$FA_{\emptyset,\mathcal{E}} ::= \mathcal{E} \mid FApp(FA_{\emptyset,\mathcal{E}}, FA_{\emptyset,\mathcal{E}}) \mid BApp(FA_{\emptyset,\mathcal{E}}, FA_{\emptyset,\mathcal{E}})$$
$$\mid FApp_\emptyset(FA_{\emptyset,\mathcal{E}}) \mid BApp_\emptyset(FA_{\emptyset,\mathcal{E}})$$

**Definition 14 ($FA_\emptyset$ structure of a $GAB_\emptyset$ deduction for $NL_\emptyset$).** *To each $GAB_\emptyset$ deduction $\mathcal{P}$ for $NL_\emptyset$, we associate a $FA_\emptyset$ structure, written $FA_{\emptyset,Tp}(\mathcal{P})$, such that each internal node corresponds to the application of a rule in $\mathcal{P}$ and is labelled by the name of this rule and where the leaves are the same as in $\mathcal{P}$.*

**Definition 15 ($GAB_\emptyset$ Deductions of $F \vdash_{GAB_\emptyset} A$ or $\Gamma \vdash_{GAB_\emptyset} A$).** *Like for GAB, if $F \in FA_{\emptyset,Tp}$ and $A \in Tp$, we say that $\mathcal{P}$ is a $GAB_\emptyset$ deduction of $F \vdash_{GAB_\emptyset} A$ when $FA_{\emptyset,Tp}(\mathcal{P}) = F$. If $\Gamma \in \mathcal{T}_{Tp}$ and $A \in Tp$, we say that $\mathcal{P}$ is a $GAB_\emptyset$ deduction of $\Gamma \vdash_{GAB_\emptyset} A$ when $A$ is the type of the conclusion of $\mathcal{P}$ and when $tree_{Tp}(FA_{\emptyset,Tp}(\mathcal{P})) = \Gamma$.*[8]

***$GAB_\emptyset$* Languages** We can associate to each categorial grammar a language of $FA_\emptyset$ structures in a similar way as for *NL*. We write respectively $\mathcal{FL}_{\emptyset GAB_\emptyset}(G)$, $\mathcal{BL}_{GAB_\emptyset}(G)$, and $\mathcal{L}_{GAB_\emptyset}(G)$, the language of $FA_\emptyset$ structures, the language of well-bracketed lists of words and the language of strings corresponding to $G$.

*Example 4.* With the categorial grammar that is defined in Example 2, we get:

$$John\ likes\ Mary \in \mathcal{L}_{GAB_\emptyset}(G_2)\ ;\ John\ likes\ little\ Mary \in \mathcal{L}_{GAB_\emptyset}(G_2)$$

because we can build the following deductions:

$$\dfrac{\dfrac{\dfrac{\overbrace{(L\backslash L)\backslash N}^{John}}{N}\,BApp_\emptyset \quad \overbrace{N\backslash(S/N)}^{likes}}{S/N}\,BApp \quad \dfrac{\overbrace{(L\backslash L)\backslash N}^{Mary}}{N}\,BApp_\emptyset}{S}\,FApp \qquad and \qquad \dfrac{\dfrac{\dfrac{\overbrace{(L\backslash L)\backslash N}^{John}}{N}\,BApp_\emptyset \quad \overbrace{N\backslash(S/N)}^{likes}}{S/N}\,BApp \quad \dfrac{\overbrace{L\backslash L}^{little} \quad \overbrace{(L\backslash L)\backslash N}^{Mary}}{N}\,BApp}{S}\,FApp$$

[8] $tree_{Tp}$ erases unary combinators $FApp_\emptyset$ and $BApp_\emptyset$ during translation: $tree_{Tp}(FApp_\emptyset(F)) = tree_{Tp}(BApp_\emptyset(F)) = tree_{Tp}(F)$

***NL*$_\emptyset$ and *GAB*$_\emptyset$ languages** In fact, there is a strong correspondence between $GAB_\emptyset$ deductions and $NL_\emptyset$ derivations. In particular with Theorem 3, we show that the respective string languages and binary tree languages are the same.

**Theorem 3.** *If $A$ is an atomic formula, $\Gamma \vdash_{GAB_\emptyset} A$ iff $\Gamma \vdash_{NL_\emptyset} A$*

**Corollary 2.** *$\mathcal{BL}_{NL_\emptyset}(G) = \mathcal{BL}_{GAB_\emptyset}(G)$ and $\mathcal{L}_{NL_\emptyset}(G) = \mathcal{L}_{GAB_\emptyset}(G)$*

We write, for the rest of the paper, $\mathcal{FL}_\emptyset(G)$, $\mathcal{BL}_\emptyset(G)$ and $\mathcal{L}_\emptyset(G)$ in place of $\mathcal{FL}_{GAB_\emptyset}(G)$, $\mathcal{BL}_{GAB_\emptyset}(G) = \mathcal{BL}_{NL_\emptyset}(G)$ and $\mathcal{L}_{GAB_\emptyset}(G) = \mathcal{L}_{NL_\emptyset}(G)$.

**Proof of $\Gamma \vdash_{GAB_\emptyset} A \Rightarrow \Gamma \vdash_{NL_\emptyset} A$ ($A$ does not need to be atomic) :** This is relatively easy because a $GAB_\emptyset$ deduction is just a mixed presentation of an $NL_\emptyset$ proof using a natural deduction part and a $NL_\emptyset$ derivation part (hypotheses on nodes). We can transform recursively a $GAB_\emptyset$ deduction. Suppose that the last rule of a $GAB_\emptyset$ deduction corresponding to a $FA_\emptyset$ structure $FApp_\emptyset(F)$ is:

$$\dfrac{\begin{array}{c}\mathcal{P}\\ \vdots \\ A/B\end{array}}{A}\ FApp_\emptyset$$

We know that $\vdash_{NL_\emptyset} B$ and we have a sub-deduction $\mathcal{P}$ that corresponds to $F$. $\mathcal{P}$ concludes with $A/B$. By induction hypothesis, the deduction corresponds to a $NL_\emptyset$ derivation of $tree_{Tp}(F) \vdash_{NL_\emptyset} A/B$.

Now, using two axioms and $(/L)$ for proving $(A/B, B) \vdash A$ and two cuts, we find that $tree_{Tp}(FApp_\emptyset(F)) = tree_{Tp}(F) \vdash_{NL_\emptyset} A$. The other possibilities ($(BApp_\emptyset)$, $(FApp)$ or $(BApp)$ as first rule) are very similar and the base case is obvious.

**Proof of $\Gamma \vdash_{NL_\emptyset} A \Rightarrow \Gamma \vdash_{GAB_\emptyset} A$ ($A$ atomic) :** This property results from an adaptation to $NL_\emptyset$ of the alternative presentation of $NL$ where contexts are in a limited form [9]. This presentation, as far as we know, is original. The proof of the equivalence of $NL_\emptyset$ and this system that we call $NL^*_\emptyset$ is given as an appendix to the paper:

$$\frac{}{A \vdash A}\ \mathbf{Ax} \qquad \frac{(C, B) \vdash A}{C \vdash A/B}\ /R^* \qquad \frac{(A, C) \vdash B}{C \vdash A\backslash B}\ \backslash R^*$$

$$\frac{B \vdash A}{\emptyset \vdash A/B}\ /R^*_\emptyset \qquad \frac{A \vdash B}{\emptyset \vdash A\backslash B}\ \backslash R^*_\emptyset$$

$$\frac{D \vdash C \quad \Delta[B] \vdash A}{\Delta[(B/C, D)] \vdash A}\ /L^* \qquad \frac{D \vdash C \quad \Delta[B] \vdash A}{\Delta[(D, C\backslash B)] \vdash A}\ \backslash L^*$$

$$\frac{\emptyset \vdash C \quad \Delta[B] \vdash A}{\Delta[B/C] \vdash A}\ /L^*_\emptyset \qquad \frac{\emptyset \vdash C \quad \Delta[B] \vdash A}{\Delta[C\backslash B] \vdash A}\ \backslash L^*_\emptyset$$

Draft

If we have a derivation of $\Gamma \vdash_{NL_\emptyset} A$ with $A$ atomic using this presentation, the first rule on the main branch of the derivation must be a left rule. For instance, for $(/L^*_\emptyset)$, $\Gamma$ can be written $\Delta[B/C]$ and we have a derivation of $\vdash C$ and another one of $\Delta[B] \vdash A$. We can apply our hypothesis to the second derivation. At this point, we have a $GAB_\emptyset$ deduction $\mathcal{P}[B]$ of $\Delta[B] \vdash_{GAB_\emptyset} A$. In this deduction, we replace the leaf node corresponding to $B$ by a node corresponding to the conclusion of $(FApp_\emptyset)$ rule:

$$\begin{array}{c} B \\ \vdots \\ \mathcal{P} \end{array} \rightarrow \begin{array}{c} \dfrac{B/C}{B}\, FApp_\emptyset \\ \vdots \\ \mathcal{P} \end{array}$$

This transformation gives a $GAB_\emptyset$ deduction corresponding to $\Delta[B/C]$ since $\vdash_{NL_\emptyset} C$. The other three possibilities for $(\backslash L^*_\emptyset)$, $(/L^*)$ or $(\backslash L^*)$ are similar and give respectively a new $BApp_\emptyset$, $FApp$ or $BApp$ node and the base case where the derivation is an axiom is obvious. ■

### 5.2 A strict hierarchy theorem for $NL_\emptyset$

For each $k \in \mathbb{N}$, we consider the class $\mathcal{C}^k_{NL_\emptyset}$ of languages corresponding to $k$-valued $NL_\emptyset$ Lambek grammars (without product). This section proves that this family also forms a strict hierarchy (if the lexicon has at least 2 elements):

**Theorem 4.** $\forall k \in \mathbb{N} \quad \mathcal{C}^k_{NL_\emptyset} \subsetneq \mathcal{C}^{k+1}_{NL_\emptyset}$

### 5.3 Details of proof

We consider the same languages $L_{NL,k}$ as for $NL$.

- We first show that $L_{NL,k}$ is also $k+1$-valued for $NL_\emptyset$ using the same grammars $\sigma(G_k)$ as for $NL$ and we show $\mathcal{L}_{NL_\emptyset}(\sigma(G_k)) = L_{NL,k}$.
  - the fact that $L_{NL,k} \subseteq \mathcal{L}_{NL_\emptyset}(\sigma(G_k))$ follows from the property that a sequent valid in $NL$ is also valid in $NL_\emptyset$ ;
  - the converse is obtained by the same powerset residuated semi-groupoid (with unit) as for $NL$ (a model of both systems).
- We then have to show that $L_{NL,k}$ is not $k$-valued for $NL_\emptyset$.
  We proceed similarly as for NL, with the observation that in step 4.(b), $A_i$ does give two subtrees with one $b$ and is deduced by FApp or BApp, since if $A$ was deduced by FAppvide or BAppvide, this would contradict the assumption that $A_i$ is the smallest subtree with a yield including both $b$.

### 5.4 A proof revisited for AB (classical)

We consider again the same languages $L_{NL,k}$ as for $NL$ and show how to recover a hierarchy theorem for the classical system using the former construction.

- We first show that $L_{NL,k}$ is also $k+1$-valued for $AB$ using the same grammars $\sigma(G_k)$ as for $AB$ and we show $\mathcal{L}_{AB}(\sigma(G_k)) = L_{NL,k}$.
  - the fact that $L_{NL,k} \subseteq \mathcal{L}_{AB}(\sigma(G_k))$ follows from the derivations schemas shown for $NL$ that only involve AB rules. [9]

[9] An alternate justification is to start from $L_{AB,k} = \mathcal{L}(G_k)$, we get by substitution $L_{AB,k} \subseteq \mathcal{L}(\sigma G_k)$ and finally consider the derivation for $abb$ ($L_{NL,k} = L_{AB,k} \cup \{abb\}$).

- the converse is deduced from the fact that a AB deduction is also a $NL$ deduction: $\mathcal{L}_{AB}(\sigma(G_k)) \subseteq \mathcal{L}_{NL}(\sigma(G_k)) = L_{NL,k}$.

– We then have to show that $L_{NL,k}$ is not $k$-valued for $AB$.
We proceed similarly as for $NL$ where the derivations conditions $C_i \vdash D_i$ are replaced by $C_i = D_i$.

## 6 Conclusion

The paper studies two related lexicalized grammatical systems: non-associative Lambek grammars without product with ($NL_\emptyset$) and without ($NL$) antecedent. For each system, we prove 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.

A second important contribution of the paper consists in defining a system of generalized AB deductions and their corresponding generalized functor-argument structures for $NL_\emptyset$ (without product). This construction enables us to define languages of structured sentences as for classical categorial grammars or for $NL$ languages.

The result can not be adapted directly to other systems like non-associative Lambek calculus with product or associative Lambek calculus because the proofs depend on the existence of generalized AB deductions for both systems studied here and we do not know how to define such structures for those logical systems. Thus the questions of a strict hierarchy of languages for $k$-valued grammars are still open for them.

## References


1. Bar-Hillel, Y.: A quasi arithmetical notation for syntactic description. Language **29** (1953) 47–58
2. Lambek, J.: The mathematics of sentence structure. American mathematical monthly **65** (1958) 154–169
3. Joshi, A.K., Shabes, Y.: Tree-adjoining grammars and lexicalized grammars. In: Tree Automata and LGS. Elsevier Science, Amsterdam (1992)
4. Gold, E.: Language identification in the limit. Information and control **10** (1967) 447–474
5. 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)
6. Béchet, D., Foret, A.: k-valued non-associative lambek grammars are learnable from function-argument structures. In de Queiroz, R., Pimentel, E., Figueiredo, L., eds.: Electronic Notes in Theoretical Computer Science. Volume 84., Elsevier (2003) 1–13
7. Lambek, J.: On the calculus of syntactic types. In Jakobson, R., ed.: Structure of language and its mathematical aspects. American Mathematical Society (1961) 166–178


Draft

8. Kandulski, M.: The non-associative lambek calculus. In W. Buszkowski, W.M., Bentem, J.V., eds.: Categorial Grammar. Benjamins, Amsterdam (1988) 141–152
9. Aarts, E., Trautwein, K.: Non-associative Lambek categorial grammar in polynomial time. Mathematical Logic Quaterly **41** (1995) 476–484
10. Buszkowski, W.: Mathematical linguistics and proof theory. [14] chapter 12 683–736
11. Moortgat, M.: Categorial type logic. [14] chapter 2 93–177
12. de Groote, P.: Non-associative Lambek calculus in polynomial time. In: $8^th$ Workshop on theorem proving with analytic tableaux and related methods. Number 1617 in Lecture Notes in Artificial Intelligence, Springer-Verlag (1999) 128–139
13. de Groote, P., Lamarche, F.: Classical non-associative lambek calculus. Studia Logica **71(3)** (2002) 355–388
14. van Benthem, J., ter Meulen, A., eds.: Handbook of Logic and Language. North-Holland Elsevier, Amsterdam (1997)

## A Proof of the equivalence of $NL_\emptyset$ and $NL^*_\emptyset$

**Lemma 1.** *$NL_\emptyset$ and $NL^*_\emptyset$ are two equivalent logical systems*

**Proof.** Of course because $NL^*_\emptyset$ rules are restrictions of $NL_\emptyset$ rules, a proof of a sequent $\Gamma \vdash A$ in $NL^*_\emptyset$ is also a proof in $NL_\emptyset$. For the reverse implication, since we can eliminate cut in $NL_\emptyset$ derivations, we suppose that we have only cut free $NL_\emptyset$ derivations. In fact, we prove by induction on $k \in \mathbb{N}$ that we can rewrite a cut free $NL_\emptyset$ derivation $\mathcal{P}$ of a sequent $\Gamma \vdash A$ containing at most $k$ operators $/$ and $\backslash$ into a $NL^*_\emptyset$ derivation.

- For $k = 0$, the sequent is $B \vdash A$ with $A$ and $B$ primitive types. $\mathcal{P}$ can only be an axiom and $A = B$. Thus we have a $NL^*_\emptyset$ derivation.
- If the last rule $\mathcal{P}$ is an axiom, we already have a $NL^*_\emptyset$ derivation.
- If the last rule is also a $NL^*_\emptyset$ rule, by induction, we can find a proof of the premise(s) in $NL^*_\emptyset$ (the sequents have less than $k$ connectives) and build a proof in $NL^*_\emptyset$ of the initial sequent.
- If the last rule of $\mathcal{P}$ is $(/R)$ (the case of $(\backslash R)$ is symmetrical) but is not a $NL^*_\emptyset$ rule, we have the following proof:

$$\dfrac{\begin{array}{c}\mathcal{P}_1\\ \vdots \\ (\Gamma, B_1) \vdash A_1\end{array}}{\Gamma \vdash A_1/B_1}\,/R$$

Because the rule is not a $NL^*_\emptyset$ rule, $\Gamma$ is neither $\emptyset$ nor a type. By induction, there exists a $NL^*_\emptyset$ derivation $\mathcal{P}'_1$ of $(\Gamma, B_1) \vdash A_1$. We consider the last rule of $\mathcal{P}'_1$. The possible rules are $(/L^*)$, $(/L^*_\emptyset)$, $(\backslash L^*)$ and $(\backslash L^*_\emptyset)$. The four cases are very similar and we just look here at the first one, the case where the rule is $(/L^*)$.

We have already mentioned that $\Gamma$ is neither $\emptyset$ nor a type. At this point we have the following $NL^*_\emptyset$ deduction, where $\Gamma = \Delta_2[(B_2/C_2, D_2)]$:

$$\dfrac{\begin{array}{c}\mathcal{P}_2\\ \vdots \\ D_2 \vdash C_2\end{array} \qquad \begin{array}{c}\mathcal{Q}_2\\ \vdots \\ (\Delta_2[B_2], B_1) \vdash A_1\end{array}}{(\Delta_2[(B_2/C_2, D_2)], B_1) \vdash A_1}\,/L^*$$

Now, we can consider the following $NL_\emptyset$ derivation (each $NL^*_\emptyset$ rule is considered as a $NL_\emptyset$ rule inside the derivation):

$$\dfrac{\begin{array}{c}\mathcal{Q}_2\\ \vdots\\ (\Delta_2[B_2], B_1) \vdash A_1\end{array}}{\Delta_2[B_2] \vdash A_1/B_1}\,/R$$

By induction hypothesis, there exists a $NL^*_\emptyset$ derivation $\mathcal{Q}'_2$ of $\Delta_2[B_2] \vdash A_1/B_1$ and we finally have a derivation of $\Gamma \vdash A$ that uses $NL^*_\emptyset$ rules only:

$$\dfrac{\begin{array}{cc}\mathcal{P}_2 & \mathcal{Q}'_2\\ \vdots & \vdots\\ D_2 \vdash C_2 & \Delta_2[B_2] \vdash A_1/B_1\end{array}}{\Delta_2[(B_2/C_2, D_2)] \vdash A_1/B_1}\,/L^*$$

– If the last rule of $\mathcal{P}$ is $(/L)$ (the case of $(\backslash L)$ is symmetrical) but is not a $NL^*_\emptyset$ rule, we have the following proof, where $\Gamma = \Delta_1[(B_1/C_1, \Gamma_1)]$:

$$\dfrac{\begin{array}{cc}\mathcal{P}_1 & \mathcal{Q}_1\\ \vdots & \vdots\\ \Gamma_1 \vdash C_1 & \Delta_1[B_1] \vdash A\end{array}}{\Delta_1[(B_1/C_1, \Gamma_1)] \vdash A}\,/L$$

Because the rule is not a $NL^*_\emptyset$ rule, $\Gamma_1$ is neither $\emptyset$ nor a type. By induction, there exists a $NL^*_\emptyset$ derivation $\mathcal{P}'_1$ of $\Gamma_1 \vdash C_1$. We consider the last rule of $\mathcal{P}'_1$. The possible rules are $(/L^*)$, $(/L^*_\emptyset)$, $(\backslash L^*)$ and $(\backslash L^*_\emptyset)$ ($\Gamma_1$ is neither $\emptyset$ nor a type).

The four cases are very similar and we just look here at the first one, the case where the rule is $(/L^*)$. At this point we have the following $NL^*_\emptyset$ deduction, where $\Gamma_1 = \Delta_2[(B_2/C_2, D_2)]$:

$$\dfrac{\begin{array}{cc}\mathcal{P}_2 & \mathcal{Q}_2\\ \vdots & \vdots\\ D_2 \vdash C_2 & \Delta_2[B_2] \vdash C_1\end{array}}{\Delta_2[(B_2/C_2, D_2)] \vdash C_1}\,/L^*$$

Now, we can consider the following $NL_\emptyset$ derivation (each $NL^*_\emptyset$ rule is considered as a $NL_\emptyset$ rule inside the derivation):

$$\dfrac{\begin{array}{cc}\mathcal{Q}_2 & \mathcal{Q}_1\\ \vdots & \vdots\\ \Delta_2[B_2] \vdash C_1 & \Delta_1[B_1] \vdash A\end{array}}{\Delta_1[(B_1/C_1, \Delta_2[B_2])] \vdash A}\,/L$$

By induction hypothesis, there exists a $NL^*_\emptyset$ derivation $\mathcal{Q}'_2$ of $\Delta_1[(B_1/C_1, \Delta_2[B_2])] \vdash A$ and we finally have a derivation of $\Gamma \vdash A$ that uses $NL^*_\emptyset$ rules only:

$$\dfrac{\begin{array}{cc}\mathcal{P}_2 & \mathcal{Q}'_2\\ \vdots & \vdots\\ D_2 \vdash C_2 & \Delta_1[(B_1/C_1, \Delta_2[B_2])] \vdash A\end{array}}{\Delta_1[(B_1/C_1, \Delta_2[(B_2/C_2, D_2)])] \vdash A}\,/L^*$$

■

Draft