On Intermediate Structures for Non-Associative Lambek Grammars and Learnability

Denis Béchet and Annie Foret [a,b]

[a] *LIPN – UMR 7030*
CNRS & Université Paris 13
99, avenue J.-B. Clement
93430 Villetaneuse
France

[b] *IRISA – Université de Rennes 1*
Campus Universitaire de Beaulieu
Avenue du Général Leclerc
35042 Rennes Cedex
France

Abstract

This paper is concerned with learning categorial grammars in the model of Gold. We show that rigid and k-valued non-associative Lambek (NL) grammars are not learnable from well-bracketed sentences.

In contrast to k-valued classical categorial grammars, k-valued Lambek grammars are not learnable from strings. This result was shown for several variants including the non-associative variant *NL*. Whereas k-valued Lambek *NL* grammars have been shown learnable from functor-argument structures, we show that non-learnability still holds for the languages of well-bracketed strings ; this result is obtained for two variants *NL* and $NL_\emptyset$ by exhibiting a limit point in the considered class.

Such a result aims at clarifying the possible directions for future learning algorithms: it expresses the difficulty of learning categorial grammars from strings or intermediate structures and helps to draw the frontier between learnability and non-learnability.

Key words: grammatical inference, categorial grammars, non-associative Lambek calculus, learning from positive examples, model of Gold, structured data

Email addresses: Denis.Bechet@lipn.univ-paris13.fr, Annie.Foret@irisa.fr (Denis Béchet and Annie Foret).

1 Introduction

Categorial grammars [1] and Lambek grammars [2,3] have been studied in the field of natural language processing. They are well adapted to learning perspectives since they are completely lexicalized and an actual way of research is to determine the sub-classes of such grammars that remain learnable in the sense of Gold [4].

We recall that learning here consists to define an algorithm on a finite set of sentences that converges to obtain a grammar in the class that generates the examples. Let $\mathcal{G}$ be a class of grammars, that we wish to learn from positive examples. Formally, let $\mathcal{L}(G)$ denote the language associated with grammar G, and let V be a given alphabet, a learning algorithm is a function ϕ from finite sets of words in V^* to $\mathcal{G}$, such that for all $G \in \mathcal{G}$ with $\mathcal{L}(G) =< e_i >_{i\in\mathbb{N}}$ there exists a grammar $G' \in \mathcal{G}$ and there exists $n_0 \in \mathbb{N}$ such that: $\forall n > n_0$ $\phi(\{e_1, \ldots, e_n\}) = G' \in \mathcal{G}$ with $\mathcal{L}(G') = \mathcal{L}(G)$.

After pessimistic unlearnability results in [4], learnability of non trivial classes has been proved in [5] and [6]. Recent works from [7] and [8] following [9] have answered the problem for different sub-classes of classical categorial grammars (we recall that the whole class of classical categorial grammars is equivalent to context free grammars; the same holds for the class of Lambek grammars [10] that is thus not learnable in Gold's model).

The extension of such results for Lambek grammars is an interesting challenge that is addressed by works on logic types from [11] (these grammars enjoy a direct link with Montague semantics), learning from structures in [12], complexity results from [13] or unlearnability results from [14,15]; on the one hand, this unlearnability result was shown for several variants including the non-associative variant NL in the case of string languages ; on the other hand k-valued Lambek NL grammars have been shown learnable from functor-argument structures [16,17].

In this paper, we consider the following question: is the non-associative variant NL of k-valued Lambek grammars learnable from well-bracketed strings ; we answer for both variants NL and $NL_\emptyset$ by providing a limit point for this class.

The paper is organized as follows. Section 2 gives some background knowledge on three main aspects: non-associative Lambek categorial grammars ; learning from posivite data (in Gold's model) ; some useful models of Lambek calculus. Section 3 gives some review on learning string languages. Section 4 and 5 then present our main results on NL and $NL_\emptyset$ (this latter denotes non-associative Lambek grammars allowing empty sequence): after a construction overview, we discuss some corollaries and then provide the details of proof. 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 [2,3] or more recently in [18–23]. We use in the paper non-associative Lambek calculus with and without empty sequence ($NL_\emptyset$ and NL) but without product.

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

- *Σ is a finite alphabet (the words in the sentences);*
- *$I : \Sigma \mapsto \mathcal{P}^f(T)$ is a function that maps a 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 $X \in I(a)$, 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 Calculus NL

2.2.1 NL Derivation $\vdash_{NL}$

As a logical system, we use Gentzen-style sequent presentation. A sequent $\Gamma \vdash A$ is composed of a binary tree of formulas Γ (the set of 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.

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

[1] no product connective is used in the paper

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

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

We write $NL_\emptyset$ for the non-associative Lambek calculus allowing empty antecedents (the left part Γ of the sequent may be empty). Here, we use two identity relations:

$$\Gamma[(\emptyset, \Delta)] \equiv \Gamma[\Delta] \equiv \Gamma[(\Delta, \emptyset)]$$

Theorem 4 (Cut elimination) *We recall that the cut rule can be eliminated in $\vdash_{NL}$: every derivable sequent has a cut-free derivation.*

2.2.2 NL Languages

Definition 5 (Yield) *If T is a binary 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 strings [2] $yield_\Sigma$ and for binary trees of formulas $yield_{Tp}$.

Definition 6 (Language) *Let $G = (\Sigma, I, S)$ be a categorial grammar.*

- *G* generates a well-bracketed string *$T \in \mathcal{T}_\Sigma$ (in NL model) iff there exist Γ a binary tree of types, $c_1, \ldots, c_n \in \Sigma$ and $A_1, \ldots, A_n \in Tp$ such that:*
$$\begin{cases} G : c_i \mapsto A_i\ (1 \leq i \leq n) \\ \Gamma = T[c_1 \to A_1, \ldots, c_n \to A_n] \\ \Gamma \vdash_{NL} S \end{cases}$$
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 strings *corresponding to G, written $\mathcal{BL}_{NL}(G)$, is the set of well-bracketed strings generated by G.*
- *The* language of strings *corresponding to G, written $\mathcal{L}_{NL}(G)$, is the set of strings generated by G.*
- *We use similar definitions for $NL_\emptyset$ instead of NL.*

[2] in the paper, a well-bracketed string is a binary tree of words.

For a class $\mathcal{G}$ of grammars, we write $\mathcal{L}(G)$ the language that is generated by $G \in \mathcal{G}$ and $\mathcal{L}(\mathcal{G}) = \{\mathcal{L}(G) \mid G \in \mathcal{G}\}$ the class of generated languages.

Example 1 *Let $\Sigma_1 = \{John, Mary, likes\}$ and let $Pr_1 = \{S, N\}$. We define:*

$$G_1 = \begin{cases} John \mapsto N \\ Mary \mapsto N \\ likes \mapsto N\backslash(S/N) \end{cases}$$

G_1 is a rigid (or 1-valued) grammar. We can prove that $((N,\ N\backslash(S/N)),\ N) \vdash_{NL} S$. Thus, we get:

$$John\ likes\ Mary \in \mathcal{L}_{NL}(G_1)$$

$$((John\ likes)\ Mary) \in \mathcal{BL}_{NL}(G_1)$$

2.3 Learning and Limit Points

We now recall some useful definitions and known properties on learning.

Definition 7 (Limit Points) *A class $\mathcal{CL}$ of languages has* a limit point *iff there exists an infinite sequence $< L_n >_{n\in N}$ of languages in $\mathcal{CL}$ and a language $L \in \mathcal{CL}$ such that: $L_0 \subsetneq L_1 \ldots \subsetneq ... \subsetneq L_n \subsetneq \ldots$ and $L = \bigcup_{n\in N} L_n$ (L is* a limit point *of $\mathcal{CL}$).*

Property 8 (Limit Points Imply Unlearnability) *If the languages of the grammars in a class $\mathcal{G}$ have a limit point then the class $\mathcal{G}$ is* unlearnable.[3]

2.4 Some Useful Models

For ease of proof involving limit points, models are often helpful ; for example in [24] free groups and recent free pregroups [25] are used. For bracketed strings in *NL*, we use another kind of model : Quasi-groups ; such models have been used for categorial grammars in [26–28].

2.4.1 Quasi-groups

Quasi-groups are sets equiped with three binary operations $., /, \backslash$ that satisfy the following equations :

[3] This implies that the class has infinite elasticity. A class $\mathcal{CL}$ of languages has *infinite elasticity* iff $\exists < e_i >_{i\in\mathbb{N}}$ sentences $\exists < L_i >_{i\in\mathbb{N}}$ languages in $\mathcal{CL}$ $\forall i \in \mathbb{N} : e_i \notin L_i$ and $\{e_1, \ldots, e_n\} \subseteq L_{n+1}$.

$x.(x\backslash y) = y$

$(x/y).y = x$

$x\backslash(x.y) = y$

$(x.y)/y = x$

The equational theory, here written QG, of quasi-groups admits a canonical rewriting system as found by Knuth and Bendix (70) and further studied in [29]. The case of quasi-groups with a neutral element e is similar, see [29] for details. [4]

The above four equations oriented from left to right are completed with two new rules to produce such a canonical rewrite system :

$(x/y)\backslash x \rightarrow y$

$x/(y\backslash x) \rightarrow y$

2.4.2 Quasi-groups as models

Quasi-groups are models for $\vdash_{NL}$.

From types to QG-formulas. We associate with each type formula A an element in quasi-groups written $[\![A]\!]$ as follows [5] :

$[\![p]\!] = p$ for p atomic

$[\![A_1\backslash A_2]\!] = [\![A_1]\!]\backslash[\![A_2]\!]$

$[\![A_1/A_2]\!] = [\![A_1]\!]/[\![A_2]\!]$

We extend the notation to binary trees by :

$[\![(A_1, A_2)]\!] = [\![A_1]\!].[\![A_2]\!]$

Proposition 1 (QG-Models) *If* $\Gamma \vdash_{NL} A$ *then* $[\![\Gamma]\!] =_{QG} [\![A]\!]$

Quasi-groups may also be seen as *residuated groupoids* that are known as (complete) algebraic models [20] for NL : take equality as partial ordering.

---

[4] one adds the equations : $e.x = x \quad x.e = x$
that are also oriented from left to right and a canonical rewrite system is obtained by adding the following rules : $e\backslash x \rightarrow x \quad x/e \rightarrow x \quad x/x \rightarrow e \quad x\backslash x \rightarrow e$

[5] for a calculus with product $\otimes$, we add $[\![A_1 \otimes A_2]\!] = [\![A_1]\!].[\![A_2]\!]$.

3 Limit Point Construction for *NL*

We now cite some results from [24], where the authors have given a limit point for languages of strings. We shall use this construction in next sections.

Form of grammars. We consider grammars (as given in [24]) G_n where A, B, D_n and E_n are complex types and S is the main type of each grammar :

$$G_n = \{a \mapsto A/B \ ; \ b \mapsto D_n \ ; \ c \mapsto E_n \backslash S\}$$

Definition 9 (G_n and G_* [24]) *Let p, q, S be three primitive types :*

$$A = D_0 = E_0 = q/(p\backslash q)$$
$$B = p$$
$$D_{n+1} = (A/B)\backslash D_n$$
$$E_{n+1} = (A/A)\backslash E_n$$
$$\text{Let } G_n = \left\{ \begin{array}{l} a \mapsto A/B = (q/(p\backslash q))/p \ ; \\ b \mapsto D_n \ ; \\ c \mapsto E_n \backslash S \end{array} \right\}$$
$$\text{Let } G_* = \{a \mapsto p/p \ ; \ b \mapsto p \ ; \ c \mapsto p\backslash S\}$$

Proposition 2 (language description [24])

- $\mathcal{L}(G_n) = \{a^k bc \mid 0 \leq k \leq n\}$
- $\mathcal{L}(G_*) = \{a^k bc \mid 0 \leq k\}$.

This construction defines a limit point with the following result.

Proposition 3 (*NL*-non-learnability [24]) *The class of languages of rigid (or k-valued for an arbitrary k) non-associative Lambek grammars (not allowing empty sequence and without product) admits a limit point ; the class of rigid (or k-valued for an arbitrary k) non-associative Lambek grammars (not allowing empty sequence and without product) is not learnable from strings.*

4 Structure Languages and Limit Points.

Definition. We recall that $\mathcal{BL}_{NL}(G)$ denotes the set of well-bracketed strings generated by G in *NL*. We use a similar definition and notation for $NL_\emptyset$.

Structure languages for *NL*. We focus on a limit point construction already given for *NL* in the case of strings. To deal with well-bracketed strings, we interpret the types in quasi-groups ; we get the same images for all grammars

in the sequence. Observe that in contrast to groups, quasi-groups are here helpful due to their non-associativity and some non-simplifications to unit (for example p/p does not reduce to unit). Moreover free quasi-groups enjoy a complete rewriting system on which we may base a crucial part of reasoning.

By an easy calculus (using type-raising rules) we compute QG-normal forms :

Grammars and types	Languages	QG-normal form
$G_n = \begin{cases} a \mapsto A/B = (q/(p\backslash q))/p \\ b \mapsto D_n \\ c \mapsto E_n\backslash S \end{cases}$	$\{a^k bc \mid 0 \leq k \leq n\}$	$G_n : \begin{cases} [\![A/B]\!] = p/p \\ [\![D_n]\!] = p \\ [\![E_n\backslash S]\!] = p\backslash S \end{cases}$
$G_* = \{a \mapsto p/p \,;\, b \mapsto p \,;\, c \mapsto p\backslash S\}$	a^*bc	
where $\begin{cases} A = D_0 = E_0 = q/(p\backslash q) \\ B = p \\ D_{n+1} = (A/B)\backslash D_n \\ E_{n+1} = (A/A)\backslash E_n \end{cases}$		$G_* : \begin{cases} [\![p/p]\!] = p/p \\ [\![p]\!] = p \\ [\![p\backslash S]\!] = p\backslash S \end{cases}$

Theorem 10 *For $n \in \mathbb{N}$*

$$\mathcal{BL}_{NL}(G_*) = \{(\underbrace{a \ldots (a(a}_{k} b))) \ldots)c) \mid 0 \leq k\}$$
$$\mathcal{BL}_{NL}(G_n) = \{(\underbrace{a \ldots (a(a}_{k} b))) \ldots)c) \mid 0 \leq k \leq n\} \subsetneq \mathcal{BL}_{NL}(G_*)$$

Proof :

- We know from [24] that :
  (1) $\mathcal{L}_{NL}(G_n) = \{a^k bc \mid 0 \leq k \leq n\}$ and $\mathcal{L}_{NL}(G_*) = a^*bc$
- Let $\mathcal{BL}_* = \{(\underbrace{a \ldots (a(a}_{k} b))) \ldots)c) \mid 0 \leq k\}$
- We show by QG-models that :
  (2) $\mathcal{BL}_{NL}(G_n) \subseteq \mathcal{BL}_*$ and
  (3) $\mathcal{BL}_{NL}(G_*) \subseteq \mathcal{BL}_*$
  Since the types in G_n, or G_* have the same quasi-group images for all n, we may group the reasoning for all n in the proofs of (2) and (3).
  Let $w = a^k bc$ for some k, we consider the quasi-group image for its possible bracketed types ; it is a corresponding parenthesized version of :
  $p/p \,.\, p/p \,.\, \ldots p \,.\, p\backslash S$ the QG-normal form of which must be S.
  – In the case of no a in w : $p \,.\, p\backslash S$ corresponds to bc.
  – In the case of one a : the bracketing $p/p \,.\, (p \,.\, p\backslash S)$ is not possible since it blocks after the first rewriting step $p/p \,.\, S$ therefore, we have the following brackets : $(p/p \,.\, p) \,.\, p\backslash S$ as desired.
  – In the case of k ($k \neq 0$) occurrences of a : we consider the bracket for the

first reduction step ; if we had $\underbrace{p/p \,.\, \ldots p/p}_{k} \,.\, (p \,.\, p\backslash S)$ it would block after the first rewriting step as follows $\underbrace{p/p \,.\, \ldots p/p}_{k} \,.\, S$; therefore, we have the following brackets : $\underbrace{p/p \,.\, \ldots (p/p}_{k} \,.\, p) \,.\, p\backslash S$; with the first rewriting step $\underbrace{p/p \,.\, \ldots}_{k-1} p \,.\, p\backslash S$; we now apply the induction hypothesis for $k-1$ and get the desired parenthesizing ; namely the only possible parenthesizing for w is $(\underbrace{a \ldots (a(a}_{k} b))) \ldots)c)$

- We thus get, since each word in $\mathcal{BL}_{NL}(G_n)$ or in $\mathcal{BL}_{NL}(G_*)$ corresponds to at least one parenthesizing and (as we have just seen) it must belong to $\mathcal{BL}_*$:
  $\mathcal{BL}_{NL}(G_n) = \{(\underbrace{a \ldots (a(a}_{k} b))) \ldots)c) \mid 0 \leq k \leq n\}$
  $\mathcal{BL}_{NL}(G_*) = \{(\underbrace{a \ldots (a(a}_{k} b))) \ldots)c) \mid 0 \leq k\}$ ■

This construction defines a limit point for the structure languages in the rigid case, and more generally in the k-valued case for each k, the corresponding classes of grammars are thus not learnable from bracketed examples as stated below.

Corollary 11 *The class of well-bracketed strings for NL has a* rigid *limit point ; the class of rigid (or k-valued for an arbitrary k) non-associative NL Lambek rigid grammars is not learnable from well-bracketed strings (in Gold's model).*

5 Limit Point Construction for $NL_\emptyset$

5.1 Limit Point for $NL_\emptyset$

In the last section, we have proved that the *NL* languages of well-bracketed strings have a limit point and thus is not learnable. This proof does not work for $NL_\emptyset$ because the interpretation in the free quasi-group with unit of the types that appear in the grammars is different and does not prove that the languages corresponding to the grammars $G_n, n \in \mathbb{N}$ and G_* are included in $\{((\underbrace{a \cdots (a(a}_{k} b)) \cdots)c) \mid k \geq 0\}$.

For this system, we need another proof. In the section, we prove and use syntactical properties on sequent calculus and conclude that there is no other well-bracketed string corresponding to the grammars $G_n, n \in \mathbb{N}$ and G_*.

Lemma 12 *For any environment $\Gamma[\cdot]$, for any formula A and for any atomic formula p, $\Gamma[(p/p, p/p)] \vdash_{NL_\emptyset} A$ is not provable.*

Proof : By induction on the size of a hypothetical proof (without cut) in sequent calculus. We look at the last rule used to prove $\Gamma[(p/p, p/p)] \vdash_{NL_\emptyset} A$. If $(p/p, p/p)$ appears in the left environment of one of the premises of the rule, we can apply the induction principle and find a contradiction. If $(p/p, p/p)$ does not appear in the left environment of all the premises, this rule must be $/L$. Because, we use $NL_\emptyset$, there are three possibilities :

$$\frac{p/p \vdash p \quad \Gamma[p] \vdash A}{\Gamma[(p/p, p/p)] \vdash A}\,/L \quad \frac{\emptyset \vdash p \quad \Gamma[(p/p, p)] \vdash A}{\Gamma[(p/p, p/p)] \vdash A}\,/L \quad \frac{\emptyset \vdash p \quad \Gamma[(p, p/p)] \vdash A}{\Gamma[(p/p, p/p)] \vdash A}\,/L$$

But $p/p \vdash p$ and $\emptyset \vdash p$ are not provable in $NL_\emptyset$ (there is an odd number of atomic formulas). Thus the three possibilities are not possible ■

Lemma 13 *For any environment Γ, for any formula A and for any atomic formulas p and q, $\Gamma[((q/(p\backslash q))/p, (q/(p\backslash q))/p) \vdash_{NL_\emptyset} A$ is not provable.*

Proof : Like the previous lemma, we prove by induction on the size of a hypothetical derivation of such a sequent. Using the same idea, we have to prove that the following sequents are not provable in $NL_\emptyset$:

$$\emptyset \vdash p \qquad (q/(p\backslash q))/p \vdash p$$

It is not possible because the number of occurrences of p is odd ■

Lemma 14 *A well-bracketed string in the languages corresponding to $G_n, n \in \mathbb{N}$ and G_*, cannot contain the sub-expression (aa).*

Proof : This is a direct consequence of the previous lemmas because for $G_n, n \in \mathbb{N}$, the type associated to a is $(q/(p\backslash q))/p$ and for G_*, it is p/p ■

Lemma 15 *For any atomic formula S and for any environment $\Delta[\cdot]$ with no occurrence of S, a proof of the sequent $\Delta[S] \vdash_{NL_\emptyset} S$ implies that $\Delta[S] \equiv S$.*

Proof : By induction on the size of the proof. We look at the last rule of $\Delta[S] \vdash_{NL_\emptyset} S$. This can be an axiom in which case $\Delta[S] \equiv S$ or it can be $\backslash L$ or $/L$:

$$\frac{\Gamma_1 \vdash A_1 \quad \Delta_1[B_1] \vdash S}{\Delta_1[(B_1/A_1, \Gamma_1)] \vdash S}\,/L \quad \frac{\Gamma_1 \vdash A_1 \quad \Delta_1[B_1] \vdash S}{\Delta_1[(\Gamma_1, A_1\backslash B_1)] \vdash S}\,\backslash L$$

In both cases, S can not be in Γ_1 because there are only two occurrences of S that must end in an axiom. Thus, we need a proof of $\Delta_1[B_1] \vdash S$ where $\Delta_1[\cdot]$

contains one occurrence of S. By induction, $\Delta_1[B_1]$ must be equal to S which is not possible because $B_1 \neq S$ ■

Lemma 16 *For any atomic formula S and for any environment $\Delta[\cdot]$ and formula A with no occurrence of S, a proof in $NL_\emptyset$ of the sequent $\Delta[A\backslash S] \vdash_{NL_\emptyset} S$ implies that there exists an environment Γ such that $\Delta[A\backslash S] = (\Gamma, A\backslash S)$.*

Proof : As for the previous proof, we proceed by induction. We look at the last rule of $\Delta[A\backslash S] \vdash_{NL_\emptyset} S$ which can only be $\backslash L$ or $/L$:

$$\frac{\Gamma_1 \vdash A_1 \quad \Delta_1[B_1] \vdash S}{\Delta_1[(B_1/A_1, \Gamma_1)] \vdash S}\, /L \qquad \frac{\Gamma_1 \vdash A_1 \quad \Delta_1[B_1] \vdash S}{\Delta_1[(\Gamma_1, A_1\backslash B_1)] \vdash S}\, \backslash L$$

It is not possible for $A\backslash S$ to occur in Γ_1 because there are only two occurrences of S that must end in an axiom and must be in the same premise.

If $A\backslash S$ is not introduced by the rule, $A\backslash S$ appears in $\Delta_1[\cdot]$. By induction, we can infer that there exists Γ such that $\Delta_1[B_1] = (\Gamma[B_1], A\backslash S)$ and $\Delta[A\backslash S]$ is $(\Gamma[(B_1/A_1, \Gamma_1)], A\backslash S)$ or $(\Gamma[(\Gamma_1, A_1\backslash B_1)], A\backslash S)$.

If $A\backslash S$ is introduced by the rule (which must be $\backslash L$), we have :

$$\frac{\Gamma_1 \vdash A \quad \Delta_1[S] \vdash S}{\Delta_1[(\Gamma_1, A\backslash S)] \vdash S}\, \backslash L$$

We know with the previous lemma that $\Delta_1[S] \equiv S$. Thus $\Delta[A\backslash S] \equiv (\Gamma_1, A\backslash S)$ ■

Lemma 17 *A well-bracketed string in the languages corresponding to $G_n, n \in \mathbb{N}$ and G_* that has only one c must be of the form (Tc) where T is a well-bracketed string.*

Proof : This is a direct consequence of the previous lemmas because for $G_n, n \in \mathbb{N}$, S does not appear in the types associated to a and b and because the type associated to c is $(E_n\backslash S)$ or $(p\backslash S)$ where S does not appear in E_n or p ■

Theorem 18 *For $n \in \mathbb{N}$*

- $\mathcal{L}_{NL_\emptyset}(G_n) = \{((\underbrace{a \cdots (a(a}_{k}\, b)) \cdots)c) \mid 0 \leq k \leq n\}$
- $\mathcal{L}_{NL_\emptyset}(G_*) = \{((\underbrace{a \cdots (a(a}_{k}\, b)) \cdots)c) \mid 0 \leq k\}$

Proof : In [24], the authors have proved that the languages of strings corresponding to $G_n, n \in \mathbb{N}$ and G_* are :

- $\mathcal{L}_{NL_\emptyset}(G_n) = \{a^k bc \mid 0 \leq k \leq n\}$
- $\mathcal{L}_{NL_\emptyset}(G_*) = \{a^k bc \mid 0 \leq k\}$.

Thus, the language of well-bracketed strings of these grammars must correspond to these languages (through *yield*). Since it is not possible to have (aa) by Lemma 14 and it is necessary that the well-bracketed string is of the form (Tc) by lemma 17, the languages must be in $\{((\underbrace{a \cdots (a(a}_{k} b)) \cdots)c) \mid k \geq 0\}$ ■

Corollary 19 *The class of well-bracketed strings for $NL_\emptyset$ has a* rigid *limit point ; the class of rigid (or k-valued for an arbitrary k) non-associative $NL_\emptyset$ Lambek rigid grammars is not learnable from well-bracketed strings (in Gold's model).*

6 Conclusion and Remarks

Lambek grammars.

In the paper [24], the whole landscape of Lambek-like rigid grammars (or k-valued for an arbitrary k) is described as for the learnability question from strings (in Gold's model). In [16], we have shown that k-valued Lambek NL grammars are learnable from functor-argument structures. We have shown here that non-learnability still holds for the languages of well-bracketed strings ; this result is obtained for two variants NL and $NL_\emptyset$ by exhibiting a limit point in the considered class.

Non-learnability for subclasses.

The limit point is of order [6] 5 and does not use the product operator. Thus, we have the following corollaries:

- Restricted connectives: k-valued NL and $NL_\emptyset$ grammars **without product** are not learnable from well-bracketed strings.
- Restricted type order: k-valued NL and $NL_\emptyset$ grammars (without product) with types **not greater than order 5** are not learnable from well-bracketed strings.

The learnability question may still be raised for NL grammars of order lower than 5.

[6] Order is defined by :
$order(A) = 0$ if $A \in Pr$
$order(A/B) = order(B \backslash A) = max(order(B) + 1, order(A))$

Remarks. Such a result aims at clarifying the possible directions for future learning algorithms : it expresses the difficulty of learning categorial grammars from strings or intermediate structures and helps to draw the frontier between learnability and non-learnability :

Restriction on types. An interesting perspective for learnability results might be to introduce reasonable restrictions on types. From what we have seen, the order of type alone (order 1 excepted) does not seem to be an appropriate measure in that context. For instance in [17] the notion of bounded arity leads to learnability theorems.

Annotated examples. These results also indicate the necessity of using enough informations as input of learning algorithms. We have seen here that the binary tree structure is not enough. The learnability from functor-argument structures or from strings when arities are bounded suggests that it is more important to know the arity of the words than the global structure of the sentences.

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 (1958) 154–169.

[3] J. Lambek, On the calculus of syntactic types, in: R. Jakobson (Ed.), Structure of language and its mathematical aspects, American Mathematical Society, 1961, pp. 166–178.

[4] E. Gold, Language identification in the limit, Information and control 10 (1967) 447–474.

[5] D. Angluin, Inductive inference of formal languages from positive data, Information and Control 45 (1980) 117–135.

[6] T. Shinohara, Inductive inference from positive data is powerful, in: The 1990 Workshop on Computational Learning Theory, Morgan Kaufmann, San Mateo, California, 1990, pp. 97–110.

[7] M. Kanazawa, Learnable classes of categorial grammars, Studies in Logic, Language and Information, FoLLI & CSLI, 1998, distributed by Cambridge University Press.

[8] J. Nicolas, Grammatical inference as unification, Rapport de Recherche RR-3632, INRIA, http://www.inria.fr/RRRT/publications-eng.html (1999).

[9] W. Buszkowski, G. Penn, Categorial grammars determined from linguistic data by unification, Studia Logica 49 (1990) 431–454.

[10] M. Pentus, Lambek grammars are context-free, in: Logic in Computer Science, IEEE Computer Society Press, 1993.

[11] Dudau-Sofronie, Tellier, Tommasi, Learning categorial grammars from semantic types, in: 13th Amsterdam Colloquium, 2001.

[12] R. Bonato, C. Retoré, Learning rigid Lambek grammars and minimalist grammars from structured sentences, Third workshop on Learning Language in Logic, Strasbourg.

[13] C. C. Florêncio, Consistent Identification in the Limit of the Class k-valued is NP-hard, in: LACL, 2002.

[14] A. Foret, Y. Le Nir, Lambek rigid grammars are not learnable from strings, in: COLING'2002, 19th International Conference on Computational Linguistics, Taipei, Taiwan, 2002.

[15] A. Foret, Y. Le Nir, On limit points for some variants of rigid Lambek grammars, in: ICGI'2002, the 6th International Colloquium on Grammatical Inference, no. 2484 in Lecture Notes in Artificial Intelligence, Springer-Verlag, 2002, pp. 106–119.

[16] D. Béchet, A. Foret, k-valued non-associative Lambek grammars are learnable from function-argument structures, in: Proccedings of the 10th Workshop on Logic, Language, Information and Computation (WoLLIC'2003), volume 85, Electronic Notes in Theoretical Computer Science, 2003.

[17] D. Béchet, A. Foret, Apprentissage des grammaires de Lambek rigides et d'arité bornée pour le traitement automatique des langues, in: Actes de la Conférence d'APprentissage 2003 (CAP'2003), 2003, pp. 155–167.

[18] M. Kandulski, The non-associative Lambek calculus, in: W. M. W. Buszkowski, J. V. Bentem (Eds.), Categorial Grammar, Benjamins, Amsterdam, 1988, pp. 141–152.

[19] E. Aarts, K. Trautwein, Non-associative Lambek categorial grammar in polynomial time, Mathematical Logic Quaterly 41 (1995) 476–484.

[20] W. Buszkowski, Mathematical linguistics and proof theory, in: van Benthem and ter Meulen [30], Ch. 12, pp. 683–736.

[21] M. Moortgat, Categorial type logic, in: van Benthem and ter Meulen [30], Ch. 2, pp. 93–177.

[22] P. de Groote, Non-associative Lambek calculus in polynomial time, in: 8^th Workshop on theorem proving with analytic tableaux and related methods, no. 1617 in Lecture Notes in Artificial Intelligence, Springer-Verlag, 1999, pp. 128–139.

[23] P. de Groote, F. Lamarche, Classical non-associative Lambek calculus, Studia Logica 71.1 (2).

[24] D. Béchet, A. Foret, k-valued non-associative Lambek categorial grammars are not learnable from strings, in: ACL (Ed.), Proccedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003), 2003, pp. 351–358.

[25] J. Lambek, Type grammars revisited, in: A. Lecomte, F. Lamarche, G. Perrier (Eds.), Logical aspects of computational linguistics: Second International Conference, LACL '97, Nancy, France, September 22–24, 1997; selected papers, Vol. 1582, Springer-Verlag, 1999.

[26] K. Versmissen, Grammatical composition: Modes, models, modalities, Ph.D. thesis, Utrecht University (1996).

[27] A. Foret, Conjoinability and unification in Lambek categorial grammars, in: New Perspectives in Logic and Formal Linguistics, Proceedings Vth ROMA Workshop, Bulzoni Editore, Roma, 2001.

[28] A. Foret, On the computation of joins for non associative Lambek categorial grammars., in: Proceedings of the 17th International Workshop on Unification Valencia, Spain, June 8-9, (UNIF'03)., 2003.

[29] J. M. Hullot, Compilation de formes canoniques dans des théories équationelles, Ph.D. thesis, University Paris–Sud (1980).

[30] J. van Benthem, A. ter Meulen (Eds.), Handbook of Logic and Language, North-Holland Elsevier, Amsterdam, 1997.