Rigid Grammars in the Associative-Commutative Lambek Calculus are not Learnable

Christophe Costa Florêncio
UiL OTS, Faculty of Arts
Utrecht University

costa@let.uu.nl

Abstract

In (Kanazawa, 1998) it was shown that rigid Classical Categorial Grammars are learnable (in the sense of (Gold, 1967)) from strings. Surprisingly there are recent negative results for, among others, rigid associative Lambek (**L**) grammars.

In this paper the non-learnability of the class of rigid grammars in **LP** (Associative-Commutative Lambek calculus) and $\mathbf{LP}_\emptyset$ (same, but allowing the empty sequent in derivations) will be shown.

1 Introduction

The question of learnability of categorial grammar (CG) was first taken up in (Kanazawa, 1998). Categorial grammar is an example of a radically lexicalized formalism, the details of which will be discussed in Section 2. Kanazawa studied only subclasses of *Classical* Categorial Grammar, results for subclasses of Lambek grammars can be found in (Foret and Nir, 2002a), (Foret and Nir, 2002b).

The model of learnability used here is *identification in the limit from positive data* as introduced in (Gold, 1967).[1] In order to show the non-learnability of rigid **LP** and $\mathbf{LP}_\emptyset$ we construct so-called *limit points* (to be defined in Section 3) for these classes.

[1] Space restrictions do not allow a full exposition of this model. The interested reader is referred to the first two chapters of (Kanazawa, 1998).

2 The Lambek Calculus

Categorial grammar originated in (Ajdukiewicz, 1935) and was further developed in (Bar-Hillel, 1953) and (Lambek, 1958). This paper will only give a brief introduction in this field, (Casadio, 1988) or (Moortgat, 1997) offers a more comprehensive overview.

A categorial grammar is a set of assignments of *types* to *symbols* from a fixed alphabet Σ, the types are either primitives or are composed from types with the binary connectives $/, \backslash, \bullet$. Rules specify how types are to be combined to form new types. A string is said to be in the language generated by grammar G (written as $s \in \mathrm{L}(G)$, L is known as a *naming function*) iff G assigns types to the symbols in the string such that these types can be combined to derive the *distinguished type*, normally written as s or t.

Definition 1 *A* domain subtype *is a subtype that is in domain position, i.e. for the type $((A/B)/C)$ the domain subtypes are B and C. For the type $(C\backslash(B\backslash A))$ the domain subtypes are C and B.*

A range subtype *is a subtype that is in range position, i.e. for the type $((A/B)/C)$ the range subtypes are (A/B) and A.*
For the type $(C\backslash(B\backslash A))$ the range subtypes are $(B\backslash A)$ and A.[2]

[2] Note that product is ignored in this definition.

In an application $A/B, B \vdash A$ or $B, B\backslash A \vdash A$ the type B is an argument *and A/B and $B\backslash A$ are known as* functors.

In (Foret and Nir, 2002a) it was shown that rigid grammars (grammars that assign only one type to any particular symbol) in **L** are not learnable from strings. They made use of the fact that in **L** the axiom $A/A, A/A \rightarrow A/A$ (and in $\mathbf{L}_\emptyset$ the axiom $B/(A/A) \rightarrow B$) holds. These axioms cause contraction-like phenomena that allow the existence of limit points even in a class of rigid grammars. They defined rigid grammars $G_n, n \in \mathbb{N}$ and G_* such that $\mathrm{L}(G_n) = c(b^*a^*)^n$ and $\mathrm{L}(G_*) = c\{a,b\}^*$. For G_n the number of alternations between a sequence of a's and a sequence of b's, (both of unbounded length) is bounded. This approach is not readily applicable to either **LP** or $\mathbf{LP}_\emptyset$ grammars, since commutativity removes the bound on the number of alterations in $\mathrm{L}(G_n)$. Instead we exploit an assymmetry inherent in the Lifting operation.

As noted in (Lambek, 1988), Lifting is a closure operation as it enjoys the following properties (we write A^B for both $B/(A\backslash B)$ and $(B/A)\backslash B$):

$$A \rightarrow A^B,$$
$$(A^B)^B \rightarrow A^B,$$
$$\text{if } A \rightarrow C, \text{ then } A^B \rightarrow C^B.$$

Note that in general $A^B \not\rightarrow A$, which implies that, during a derivation, once an atomic type is lifted it cannot be lowered anymore.

The calculus **LP** was introduced in (van Benthem, 1986) because of its natural relation with a fragment of the lambda calculus, but there is also linguistic motivation for introducing commutativity. Also see (van Benthem, 1987).

All permutation closures of context-free languages are recognizable in **LP** (van Benthem, 1991). Also note that the languages expressible in **L** and **NL** are precisely the context-free languages (see (Pentus, 1993; Kandulski, 1988), respectively). These formalisms do not have the necessary expressive power to capture natural languages (which require at least mild context-sensitivity). Therefore more expressive variants have been proposed, for example

$$A \vdash A$$

$$[/I]\,\frac{(\Gamma, B) \vdash A}{\Gamma \vdash A/B} \qquad \frac{\Gamma \vdash A/B \qquad \Delta \vdash B}{(\Gamma, \Delta) \vdash A}\,[/E]$$

$$[\backslash I]\,\frac{(B, \Gamma) \vdash A}{\Gamma \vdash B\backslash A} \qquad \frac{\Gamma \vdash B \qquad \Delta \vdash B\backslash A}{(\Gamma, \Delta) \vdash A}\,[\backslash E]$$

$$[\bullet I]\,\frac{\Gamma \vdash A \qquad \Delta \vdash B}{(\Gamma, \Delta) \vdash A \bullet B} \qquad \frac{\Delta \vdash A \bullet B \qquad \Gamma[(A,B)] \vdash C}{\Gamma[\Delta] \vdash C}\,[\bullet E]$$

Figure 1: Sequent-style presentation of the natural deduction rules for **NL**.

$$[comm]\,\frac{(\Gamma, \Delta) \vdash A}{(\Delta, \Gamma) \vdash A} \qquad \frac{((\Gamma, \Delta), \Theta) \vdash A}{\overline{(\Gamma, (\Delta, \Theta)) \vdash A}}\,[ass]$$

Figure 2: Postulates for **LP**.

the multi-modal variant (MMCG) where applicability of postulates is controlled through the use of modal operators in the lexicon. This variant, without restrictions on postulates, is a Turing-complete system (Carpenter, 1999). Recently some restrictions on postulates have been proposed that restrict expressive power to (mild) context-sensitivity, see (Moot, 2002).

The presentation of **LP** used here is due to (Kurtonina and Moortgat, 1997), it takes **NL** (Figure 1) as the 'base logic'[3] and adds associativity and commutativity postulates (Figure 2). This facilitates some of the steps in our (syntactic) proofs, and makes the derivations more explicit.

3 The construction of a limit point

The following is taken from (Kapur, 1991):

Definition 2 ***Existence Of A Limit Point*** *A class $\mathcal{L}$ of languages is said to* have a limit point *if and only if there exists an infinite sequence $\langle L_n \rangle_{n \in \mathbb{N}}$ of languages in $\mathcal{L}$ such that*

$$L_0 \subset L_1 \subset \ldots \subset L_n \subset \ldots$$

and there exists another language L in $\mathcal{L}$ such

[3]Note that, unless otherwise stated, the empty sequent is not allowed, i.e. $\vdash A$ may not occur in any derivation. Lambek variants which allow the empty string have $\emptyset$ added as subscript, for example **NL** with empty sequent is written as $\mathbf{NL}_\emptyset$.

that

$$L = \bigcup_{n\in\mathbb{N}} L_n.$$

The language L is called a limit point *of $\mathcal{L}$.*

Lemma 3 *If* $\mathrm{L}(\mathcal{G})$ *has a limit point, then $\mathcal{G}$ is not (non-effectively) learnable.*

In other words, when a class has a limit point it is not learnable because the input to the learner can never provide enough information to justify convergence. Thus even allowing a non-computable learning function makes no difference in such a case, and establishing the existence of a limit point provides a very strong negative result.

Definition 4 *For $n = 0$, let G_n be defined as*

$$G_0: \begin{array}{lcl} \mathsf{s} & \mapsto & (s/a)/c \\ \mathsf{a} & \mapsto & a \\ \mathsf{c} & \mapsto & c \end{array}$$

and for any $n \in \mathbb{N}^+$, let G_n be defined as

$$G_n: \begin{array}{lcl} \mathsf{s} & \mapsto & (s/\underbrace{a^a \bullet a^a \ldots a^a}_{n\ times})/(a\backslash a^a) \\ \mathsf{a} & \mapsto & \underbrace{a \bullet a \ldots a}_{n\ times} \\ \mathsf{c} & \mapsto & a\backslash a^a \end{array}$$

and let G_+ be defined as

$$G_+: \begin{array}{lcl} \mathsf{s} & \mapsto & (s/a)/(c/c) \\ \mathsf{a} & \mapsto & a \\ \mathsf{c} & \mapsto & c/c. \end{array}$$

A final word on notation: $\sigma, \sigma', \tau \ldots$ denote strings, and σ^{perm} is the function that yields the set of all permutations of σ.[4] Concatenation of strings will be denoted with $+$, and $\vdash$ will be taken to mean $\vdash_{\mathbf{LP}}$ (or $\vdash_{\mathbf{LP}_\emptyset}$, depending on context).

Lemma 5 *The language generated by any G_n, $n \in \mathbb{N}$, is* $\bigcup\{\langle \mathsf{s}, \mathsf{a}, \mathsf{c}^{i+1}\rangle^{\mathrm{perm}} \mid 0 \leq i \leq n\}$.

`Proof:`

[4]We will slightly abuse this notation by letting it denote *any* permutation of σ, we trust this will not lead to confusion.

1. It is trivial to show that $\langle \mathsf{s}, \mathsf{a}, \mathsf{c}\rangle^{\mathrm{perm}} \subseteq \mathrm{L}(G_0)$.

   We prove that for any $n \in \mathbb{N}^+$, $\bigcup\{\langle \mathsf{s}, \mathsf{a}, \mathsf{c}^{i+1}\rangle^{\mathrm{perm}} \mid 0 \leq i \leq n\} \subseteq \mathrm{L}(G_n)$: Grammar G_n assigns $(s/\underbrace{a^a \bullet a^a \ldots a^a}_{n\text{ times}})/(a\backslash a^a)$ to s, and $a\backslash a^a$ to c. With right-elimination we get $\mathsf{s} \circ \mathsf{c} \vdash s/\underbrace{a^a \bullet a^a \ldots a^a}_{n\text{ times}}$ (and by commutation $\mathsf{c} \circ \mathsf{s} \vdash s/\underbrace{a^a \bullet a^a \ldots a^a}_{n\text{ times}}$).

   Grammar G_n assigns $\underbrace{a \bullet a \ldots a}_{n\text{ times}}$ to a. Now, the derivation $TreeLift =$

   $$\cfrac{\cfrac{[\mathrm{hypo}_1 \vdash a]^1 \quad [\mathrm{hypo}_2 \vdash a\backslash a]^2}{\mathrm{hypo}_1 \circ \mathrm{hypo}_2 \vdash a}\,[\backslash E]}{\mathrm{hypo}_1 \vdash a/(a\backslash a)}\,[/I]^2$$

   can be combined into derivation $TreeLift_n$ through n times dot-introduction to yield $\mathrm{hypo}_1 \circ \ldots \circ \mathrm{hypo}_n \vdash \underbrace{a^a \bullet a^a \ldots a^a}_{n\text{ times}}$. Using $TreeLift_n$ as an argument for right-elimination, with $(\mathsf{s} \circ \mathsf{c})^{\mathrm{perm}} \vdash s/\underbrace{a^a \bullet a^a \ldots a^a}_{n\text{ times}}$ as functor, we get $(\mathsf{s}\circ\mathsf{c})^{\mathrm{perm}} \circ (\mathrm{hypo}_1 \circ \ldots \circ \mathrm{hypo}_n) \vdash s$. With n times dot-elimination, the last of which takes $a \vdash \underbrace{a \bullet a \ldots a}_{n\text{ times}}$ as argument, the hypotheses 1 through n can be eliminated, yielding $(\mathsf{s} \circ \mathsf{c})^{\mathrm{perm}} \circ \mathsf{a} \vdash s$. Using commutation and association we also get $\mathsf{a} \circ (\mathsf{s} \circ \mathsf{c})^{\mathrm{perm}} \vdash s$, etc, so $\bigcup\{\langle \mathsf{s}, \mathsf{a}, \mathsf{c}^{i+1}\rangle^{\mathrm{perm}} \mid i = 0\} \subseteq \mathrm{L}(G_n)$.

   Grammar G_n assigns $a\backslash a^a$ to c, so the derivation $TreeCElim =$

   $$\cfrac{[\mathrm{hypo} \vdash a]^1 \quad \mathsf{c} \vdash a\backslash(a/(a\backslash a))}{\mathrm{hypo} \circ \mathsf{c} \vdash a/(a\backslash a)}\,[\backslash E]$$

   derives the same type as $TreeLift$ does. Since i $(0 \leq i \leq n)$ $TreeLift$ deductions can occur in a derivation for G_n, by replacing them with $TreeCElim$ we get $i+1$ times c in the yield of the complete deduction.

With application of associativity and commutativity rules the resulting sequent can be rearranged so that all hypotheses occur in one minimal subsequent (for example, $\mathrm{s} \circ (((\mathrm{hypo}_1 \circ \mathrm{c}) \circ \mathrm{hypo}_2) \circ ((\mathrm{c} \circ \mathrm{hypo}_3) \circ \mathrm{c})) \vdash s$ becomes $\mathrm{s} \circ ((\mathrm{hypo}_1 \circ (\mathrm{hypo}_2 \circ \mathrm{hypo}_3)) \circ (\mathrm{c} \circ (\mathrm{c} \circ \mathrm{c}))) \vdash s$), which can then be replaced through dot-elimination by a. Thus $(\mathrm{s} \circ \mathrm{c})^{\mathrm{perm}} \circ \mathrm{c}(i \text{ times}) \circ \mathrm{a} \vdash s$ is obtained, and any permutation of this as well, by commutativity and associativity. Thus $\bigcup\{\langle \mathrm{s}, \mathrm{a}, \mathrm{c}^{i+1}\rangle^{\mathrm{perm}} \mid 1 \leq i \leq n\} \subseteq \mathrm{L}(G_n)$, for any $n \in \mathbb{N}^+$.

Together with the result for $\mathrm{L}(G_0)$, this shows that $\bigcup\{\langle \mathrm{s}, \mathrm{a}, \mathrm{c}^{i+1}\rangle^{\mathrm{perm}} \mid 0 \leq i \leq n\} \subseteq \mathrm{L}(G_n)$, for any $n \in \mathbb{N}$.

2. It is trivial to show that $\mathrm{L}(G_0) \subseteq \langle \mathrm{s}, \mathrm{a}, \mathrm{c}\rangle^{\mathrm{perm}}$.

   We prove that for any $n \in \mathbb{N}^+$, $\mathrm{L}(G_n) \subseteq \bigcup\{\langle \mathrm{s}, \mathrm{a}, \mathrm{c}^{i+1}\rangle^{\mathrm{perm}} \mid 0 \leq i \leq n\}$: For a string σ to be included in a language generated by an **LP** grammar G, G must assign a type T_n to a symbol in σ that has s as range subtype. For any n, G_n assigns such a type only to the symbol s. Furthermore, s occurs only once, as range subtype, in this type. Hence s must occur (only) once in every sentence in $\mathrm{L}(G_n)$. All derivations for a string in $\mathrm{L}(G_{i\geq 1})$ will start with

$$\cfrac{\cfrac{\mathrm{s} \vdash (s/TD_n^1)/TD_n^2 \qquad \cfrac{\cfrac{Tree_a}{\vdots}}{\sigma \vdash TD_n^2}\,ass, comm}{\mathrm{s} \circ \sigma \vdash s/TD_n^1}\,[/E] \qquad \cfrac{Tree_b}{\sigma' \vdash TD_n^1}}{\cfrac{(\mathrm{s} \circ \sigma) \circ \sigma' \vdash s}{\cfrac{\vdots}{\sigma'' \circ \mathrm{s} \circ \sigma''' \vdash s}\,ass, comm, [\bullet E]}}\,[/E]$$

   where $\sigma + \sigma'$ is some permutation of $\sigma'' + \sigma'''$ (either σ'' or σ''' may be empty). Since T_n has as domain subtype $TD_n^2 = a\backslash(a^a)$, $Tree_a$ must yield $a\backslash(a^a)$. This tree can begin with a sequence of applications of the ass and $comm$ rules (which only makes sense if σ is not a single symbol), there are some possibilities after this:

   (a) since $G_n, n \geq 1$ assigns this type to c, $\sigma = \mathrm{c}$,

   (b) use of $[\backslash I]^1$. This implies that the type a^a is derived from the sequent one step up. This type is a range type only of TD_n^2 out of all types in $G_{n\geq 1}$. Therefore this derivation can end in $\cfrac{[\mathrm{hypo} \vdash a]^1 \quad \mathrm{c} \vdash a\backslash(a^a)}{\mathrm{hypo} \circ \mathrm{c} \vdash a^a}\,[\backslash E]$, which, as far as string language is concerned, is equivalent to 2a.[5] The type a^a can be interpreted as either $a/(a\backslash a)$ or $(a/a)\backslash a$, so more introduction rules can appear. All possibilities lead to some range subtype unique to TD_n^2 (with respect to the types found in G_n), therefore $\mathrm{c} \vdash a\backslash(a^a)$ must be in $Tree_a$. All the other types found in this tree must be introduced by hypotheses, and all the hypotheses introduced have to be eliminated within $Tree_a$, and all these cases are in fact equivalent to 2a.

Since T_n has only one other domain subtype $TD_n^1 = \underbrace{a^a \bullet a^a \ldots a^a}_{n \text{ times}}$, every sentence in $\mathrm{L}(G_n)$ must contain at least one symbol to which G_n assigns a type with a as range subtype, the only symbols that qualify are a and c. Given that there are no range subtypes TD_n^1 to be found in G_n, $Tree_b$ must be of the form[6]

$$\cfrac{\cfrac{Tree_1}{\tau_1 \vdash a^a} \qquad \cfrac{\cfrac{Tree_2}{\tau_2 \vdash a^a} \qquad \cfrac{\cfrac{Tree_{n\;1}}{\tau_{n\;1} \vdash a^a} \quad \cfrac{Tree_n}{\tau_n \vdash a^a}}{\vdots}\,[\bullet I]}{\tau_2 \circ \ldots \circ \tau_n \vdash a^a \bullet a^a \ldots a^a (n-1 \text{ times})}\,[\bullet I]}{\sigma' \vdash a^a \bullet a^a \ldots a^a (n \text{ times})}\,[\bullet I]$$

where $\sigma' = \tau_1 + \ldots + \tau_n$. Symbol a is assigned $\underbrace{a \bullet a \ldots a}_{n \text{ times}}$, using hypothetical reasoning and applying the Lifting rule n times this derives TD_n, hence it can be shown that $L' = \bigcup\{\langle \mathrm{s}, \mathrm{a}, \mathrm{c}^i\rangle^{\mathrm{perm}} \mid i = 1\}$

[5] Note however that this derivation is not in normal form as defined in (Tiede, 1998).

[6] This is actually a normal form for $Tree_b$, it could also be left-branching, for example. All the other possible configurations are equivalent, however, since **LP** is associative.

is a subset of the language. This case corresponds with all trees $Tree_1 \dots Tree_n$ being of the form $TreeLift$ where the hypothesis hypo is cancelled (together with $n-1$ other hypotheses) lower in the tree by n times application of $[\bullet I]$ where the last application has argument $\mathtt{a} \vdash \underbrace{a \bullet a \dots a}_{n \text{ times}}$.

Since $a^a = a/(a\backslash a)$ (the case $a^a = (a/a)\backslash a$ can be dealt with in similar fashion), any $Tree_i$ is either of the form

$$\dfrac{\dfrac{\dfrac{\dfrac{\dots \quad [\mathrm{r}_0 \vdash a\backslash a]^1}{\vdots}}{\tau_i' \vdash a}}{\tau_i' \circ \mathrm{r}_0 \vdash a/(a\backslash a)}\,[/I]^1}{\dfrac{\vdots}{\tau_i \vdash a/(a\backslash a)}\; ass, comm, [\bullet E]}$$

which given the type-assignments in $G_{n\geq 1}$ can only be a (non-normal form) variant of $TreeLift$, or

$$\text{symbol} \vdash a/(a\backslash a)$$

which, given the type-assignments in $G_{n\geq 1}$, is only compatible with the derivation $TreeCElim$. Using hypothetical reasoning and applying the Right Elimination rule $i \leq n$ times, we can obtain i times the type a^a. All remaining a's can be lifted to obtain n a^a's.

Thus, for any $n \in \mathbb{N}^+$, $\bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^{i+1}\rangle^{\mathrm{perm}} \mid 0 \leq i \leq n\} \subseteq \mathrm{L}(G_n)$, and with the result for $\mathrm{L}(G_0)$, it follows that for any $n \in \mathbb{N}$, $\bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^{i+1}\rangle^{\mathrm{perm}} \mid 0 \leq i \leq n\} \subseteq \mathrm{L}(G_n)$.

Taken together, 1 and 2 imply that for any $n \in \mathbb{N}$, $\mathrm{L}(G_n) = \bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^{i+1}\rangle^{\mathrm{perm}} \mid 0 \leq i \leq n\}$. □

Lemma 6 *The language generated by G_+ is $\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^+\rangle^{\mathrm{perm}}$.*

`Proof:`

1. We show that $\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^+\rangle^{\mathrm{perm}} \subseteq \mathrm{L}(G_+)$: Grammar G_+ assigns $(s/a)/(c/c)$ to $\mathtt{s}$, and c/c to $\mathtt{c}$. Since in **LP** the axiom $A/A, A/A \to A/A$ holds, it follows immediately that $\mathtt{c}\circ \dots \mathtt{c} \vdash c/c$, thus with right-elimination we get $\mathtt{s}\circ\mathtt{c}^+ \vdash s/a$. Grammar G_+ assigns a to $\mathtt{a}$, thus $(\mathtt{s}\circ\mathtt{c}^+)\circ\mathtt{a} \vdash s$. By associativity and commutativity any permutation of this sequent will also derive s, thus any string in $\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^+\rangle^{\mathrm{perm}}$ can be derived.

2. We show that $\mathrm{L}(G_+) \subseteq \langle \mathtt{s}, \mathtt{a}, \mathtt{c}^+\rangle^{\mathrm{perm}}$: For a string σ to be included in a language generated by an **LP** grammar G, G must assign a type T_+ to a symbol in σ that has s as subtype. Grammar G_+ assigns such a type only to the symbol $\mathtt{s}$. Furthermore, s occurs only once, as range subtype, in this type. Hence $\mathtt{s}$ must occur (only) once in every sentence in $\mathrm{L}(G_+)$. Since T_+ has only two domain subtypes $TD_+^1 = a$ and $TD_+^2 = c/c$, every sentence in $\mathrm{L}(G_+)$ must contain at least one symbol to which G_+ assigns a type with a as range subtype, the only symbol that qualifies is $\mathtt{a}$. Thus all derivations for a string in this language must start with

$$\dfrac{\dfrac{\dfrac{\mathtt{s} \vdash (s/a)/(c/c) \qquad \dfrac{Tree_+}{\sigma' \vdash c/c}}{\mathtt{s}\circ(\sigma') \vdash s/a}\,[/E] \qquad \mathtt{a} \vdash a}{(\mathtt{s}\circ(\sigma'))\circ\mathtt{a} \vdash s}\,[/E]}{\dfrac{\vdots}{\sigma''\circ\mathtt{s}\circ\sigma''' \vdash s}\; ass, comm, [\bullet E]}$$

where $\sigma' \circ \mathtt{a}$ is some permutation of $\sigma'' + \sigma'''$ (σ'' and σ''' may be empty).

Grammar G_+ assigns TD_+^2 as range subtypes to $\mathtt{c}$, so $Tree_+$ can simply be $\mathtt{c} \vdash c/c$. Some reflection will show that other possibilities must be of the (normal) form:

$$\dfrac{\dfrac{\mathtt{c}_1 \vdash c/c \qquad \dfrac{\mathtt{c}_2 \vdash c/c \qquad \dfrac{\dfrac{\mathtt{c}_i \vdash c/c \quad [c]^1}{\mathtt{c} \vdash c}\,[/E]}{\vdots}}{\mathtt{c}_2\circ\dots\circ\mathtt{c}_i \vdash c}\,[/E]}{\mathtt{c}_1\circ\dots\circ\mathtt{c}_i \vdash c}\,[/E]}{\mathtt{c}_1\circ\dots\circ\mathtt{c}_i \vdash c/c}\,[/I]^1$$

This shows that there must be one or more $\mathtt{c}$'s in every sentence in $\mathrm{L}(G_+)$. Thus the language generated by G_+ is $\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^+\rangle^{\mathrm{perm}}$. □

Theorem 7 *The class of rigid* **LP** *grammars has a limit point.*

Proof: From Lemma 5 it follows that the languages $\mathrm{L}(G_0) \subset \mathrm{L}(G_1) \subset \ldots$ form an infinite ascending chain.

By Lemma 6 $\mathrm{L}(G_+) = \langle \mathsf{s}, \mathsf{a}, \mathsf{c}^+\rangle^{\mathrm{perm}}$ and for any $n \in \mathbb{N}$ and $0 \leq i \leq n$, $\mathrm{L}(G_n) = \langle \mathsf{s}, \mathsf{a}, \mathsf{c}^{i+1}\rangle^{\mathrm{perm}}$, $\mathrm{L}(G_+) = \cup_{n\in\mathbb{N}}\mathrm{L}(G_n)$, thus $\mathrm{L}(G_+)$ is a limit point for the class of rigid **LP** grammars. □

Corollary 8 *The class of rigid* **LP** *grammars is not (non-effectively) learnable from strings.*

In contrast to Foret and Le Nir's results, it is still an open question whether the class of *unidirectional* rigid **LP** grammars is learnable; the class under consideration is bi-directional, but only because lifting is necessary for the construction to work.

Also note that the construction depends on the presence of introduction and elimination rules for the product, and cannot be (easily) adapted for a product-free version of **LP**.

In the case of $\mathbf{LP}_\emptyset$, i.e. **LP** allowing empty sequents, things are slightly less complicated, since the axiom $B/(A/A) \rightarrow B$ holds. Consider the following construction:

Definition 9 *For any $n \in \mathbb{N}$, let G_n be defined as*

$$G_n : \begin{array}{lcl} \mathsf{s} & \mapsto & s/\underbrace{a^a \bullet a^a \ldots a^a}_{n\ times} \\ \mathsf{a} & \mapsto & \underbrace{a \bullet a \ldots a}_{n\ times} \\ \mathsf{c} & \mapsto & a\backslash a^a \end{array}$$

and let G_* be defined as

$$G_* : \begin{array}{lcl} \mathsf{s} & \mapsto & (s/a)/(c/c) \\ \mathsf{a} & \mapsto & a \\ \mathsf{c} & \mapsto & c/c. \end{array}$$

Lemma 10 *The language generated by any G_n, $n \in \mathbb{N}$, is $\bigcup\{\langle \mathsf{s}, \mathsf{a}, \mathsf{c}^i\rangle^{\mathrm{perm}} \mid 0 \leq i \leq n\}$.*

The proof is very similar to the proof of Lemma 5.

Lemma 11 *The language generated by G_* is $\langle \mathsf{s}, \mathsf{a}, \mathsf{c}^*\rangle^{\mathrm{perm}}$.*

The proof is very similar to the proof of Lemma 6.

Theorem 12 *The class of rigid* $\mathbf{LP}_\emptyset$ *grammars has a limit point.*

The proof is similar to the proof of Theorem 7; Lemmas 10 and 11 imply the existence of a limit point.

Corollary 13 *The class of rigid* $\mathbf{LP}_\emptyset$ *grammars is not (non-effectively) learnable from strings.*

This corrolary gives an easy result for multiplicative intuitionistic linear logic (**MILL**), which is an alternative formulation of $\mathbf{LP}_\emptyset$:

Corollary 14 *The class of rigid* **MILL** *grammars is not (non-effectively) learnable from strings.*

4 Conclusion

We have shown that the classes of rigid **LP** and $\mathbf{LP}_\emptyset$ grammars have limit points and are thus not learnable from strings. These results, as well as the negative results from (Foret and Nir, 2002a) and (Foret and Nir, 2002b) are quite surprising in the light of certain general results in learnability theory. To quote (Kanazawa, 1998), page 159:

> *Placing a numerical bound on the complexity of a grammar can lead to a non-trivial learnable class.* [...] Together with Shinohara's ((Shinohara, 1990a), (Shinohara, 1990b)) earlier result [context-free grammars having at most k rules are learnable], this suggests that something like this may in fact turn out to be typical in learnability theory.

The negative results for Lambek-like systems show that this is not the case. Even placing bounds on the complexity of the types appearing in the grammar may not help: rigid **L** is not even learnable when the order of types is bounded to 2.

The most important (subclass of) **L**-variant for which the question of learnability is still open is (rigid) **NL**. Results on the strong generative capacity of **NL** can be found in (Tiede, 1999), where it is suggested that they may help in establishing learnability results.

A final thought concerns the claim in (Foret and Nir, 2002a) and (Foret and Nir, 2002b) that these results demonstrate the paucity of 'flat' strings as input for a learner. They suggest that enriched input (i.e. some kind of bracketing or additional semantic information) may overcome this problem, which is certainly an interesting approach. However, one could also take another approach to constructing learnable classes within some Lambek(like) calculus by restricting the use of postulates. The multimodal approach (see for example (Moortgat and Morrill, 1991)) offers a way of doing this in the lexicon. The viability of this approach is of course dependent on the learnability of the class of rigid **NL** grammars. Even given a positive result for this class it may prove to be very hard to find characterizations of learnable classes of grammars within the multimodal paradigm.

5 Appendix: Derivations

The following list of derivations was obtained using Grail[7], included to give a feel for the kind of derivations our construction allows.

The list exhaustively enumerates all (normal form) derivations and corresponding lambda terms for the string sac given the grammar G_2 and calculus $\mathbf{LP}_\emptyset$.

$$\dfrac{a \vdash a \bullet a \qquad \dfrac{\dfrac{\dfrac{s \vdash s/(a/(a\backslash a)) \bullet (a/(a\backslash a)) \qquad \dfrac{\dfrac{[p_2 \vdash a]^4 \quad c \vdash a\backslash(a/(a\backslash a))}{p_2 \circ c \vdash a/(a\backslash a)}\,[\backslash E] \qquad \dfrac{\dfrac{[s_1 \vdash a]^3 \quad [r_0 \vdash a\backslash a]^1}{s_1 \circ r_0 \vdash a}\,[\backslash E]}{s_1 \vdash a/(a\backslash a)}\,[/I]^1}{(p_2 \circ c) \circ s_1 \vdash (a/(a\backslash a)) \bullet (a/(a\backslash a))}\,[\bullet I]}{s \circ ((p_2 \circ c) \circ s_1) \vdash s}\,[/E]}{s \circ (s_1 \circ (p_2 \circ c)) \vdash s}\,[comm]}{s \circ ((s_1 \circ p_2) \circ c) \vdash s}\,[ass]}{s \circ (a \circ c) \vdash s}\,[\bullet E]^{3,4}$$

1. $(\mathbf{1}\ \langle(\mathbf{4}\ \pi^2\mathbf{2}), \lambda z_0.(z_0\ \pi^1\mathbf{2})\rangle)$

$$\dfrac{a \vdash a \bullet a \qquad \dfrac{\dfrac{s \vdash s/(a/(a\backslash a)) \bullet (a/(a\backslash a)) \qquad \dfrac{\dfrac{\dfrac{[s_1 \vdash a]^3 \quad [p_1 \vdash a\backslash a]^2}{s_1 \circ p_1 \vdash a}\,[\backslash E]}{s_1 \vdash a/(a\backslash a)}\,[/I]^2 \qquad \dfrac{[p_2 \vdash a]^4 \quad c \vdash a\backslash(a/(a\backslash a))}{p_2 \circ c \vdash a/(a\backslash a)}\,[\backslash E]}{s_1 \circ (p_2 \circ c) \vdash (a/(a\backslash a)) \bullet (a/(a\backslash a))}\,[\bullet I]}{s \circ (s_1 \circ (p_2 \circ c)) \vdash s}\,[/E]}{s \circ ((s_1 \circ p_2) \circ c) \vdash s}\,[ass]}{s \circ (a \circ c) \vdash s}\,[\bullet E]^{3,4}$$

2. $(\mathbf{1}\ \langle\lambda y_1.(y_1\ \pi^1\mathbf{2}), (\mathbf{4}\ \pi^2\mathbf{2})\rangle)$

$$\dfrac{a \vdash a \bullet a \qquad \dfrac{\dfrac{\dfrac{\dfrac{s \vdash s/(a/(a\backslash a)) \bullet (a/(a\backslash a)) \qquad \dfrac{\dfrac{[s_1 \vdash a]^3 \quad c \vdash a\backslash(a/(a\backslash a))}{s_1 \circ c \vdash a/(a\backslash a)}\,[\backslash E] \qquad \dfrac{\dfrac{[p_2 \vdash a]^4 \quad [r_0 \vdash a\backslash a]^1}{p_2 \circ r_0 \vdash a}\,[\backslash E]}{p_2 \vdash a/(a\backslash a)}\,[/I]^1}{(s_1 \circ c) \circ p_2 \vdash (a/(a\backslash a)) \bullet (a/(a\backslash a))}\,[\bullet I]}{s \circ ((s_1 \circ c) \circ p_2) \vdash s}\,[/E]}{s \circ (p_2 \circ (s_1 \circ c)) \vdash s}\,[comm]}{s \circ ((p_2 \circ s_1) \circ c) \vdash s}\,[ass]}{s \circ ((s_1 \circ p_2) \circ c) \vdash s}\,[comm]}{s \circ (a \circ c) \vdash s}\,[\bullet E]^{3,4}$$

3. $(\mathbf{1}\ \langle(\mathbf{4}\ \pi^1\mathbf{2}), \lambda z_0.(z_0\ \pi^2\mathbf{2})\rangle)$

$$\dfrac{a \vdash a \bullet a \qquad \dfrac{\dfrac{\dfrac{s \vdash s/(a/(a\backslash a)) \bullet (a/(a\backslash a)) \qquad \dfrac{\dfrac{\dfrac{[p_2 \vdash a]^4 \quad [p_1 \vdash a\backslash a]^2}{p_2 \circ p_1 \vdash a}\,[\backslash E]}{p_2 \vdash a/(a\backslash a)}\,[/I]^2 \qquad \dfrac{[s_1 \vdash a]^3 \quad c \vdash a\backslash(a/(a\backslash a))}{s_1 \circ c \vdash a/(a\backslash a)}\,[\backslash E]}{p_2 \circ (s_1 \circ c) \vdash (a/(a\backslash a)) \bullet (a/(a\backslash a))}\,[\bullet I]}{s \circ (p_2 \circ (s_1 \circ c)) \vdash s}\,[/E]}{s \circ ((p_2 \circ s_1) \circ c) \vdash s}\,[ass]}{s \circ ((s_1 \circ p_2) \circ c) \vdash s}\,[comm]}{s \circ (a \circ c) \vdash s}\,[\bullet E]^{3,4}$$

4. $(\mathbf{1}\ \langle\lambda y_1.(y_1\ \pi^2\mathbf{2}), (\mathbf{4}\ \pi^1\mathbf{2})\rangle)$

[7] Grail is an automated theorem prover, written by Richard Moot, designed to aid in the development and prototyping of grammar fragments for categorial logics.

References

Kasimir Ajdukiewicz. 1935. Die syntaktische Konnexität. *Stud. Philos.*, 1:1 27.

Yehoshua Bar-Hillel. 1953. A quasi-arithmetical notation for syntactic description. *Language*, 29:47 58.

Bob Carpenter. 1999. The Turing Completeness of Multimodal Categorial Grammars. In Jelle Gerbrandy, Maarten Marx, Maarten de Rijke, and Yde Venema, editors, *JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*, Vossiuspers. Amsterdam University Press, Amsterdam.

Claudia Casadio. 1988. Semantic categories and the development of categorial grammars. In Oehrle et al. (Oehrle et al., 1988), pages 95–124.

Annie Foret and Yannick Le Nir. 2002a. Lambek rigid grammars are not learnable from strings. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002), Taipei, Republic of China (Taiwan)*. Morgan Kaufmann Publishers and ACL.

Annie Foret and Yannick Le Nir. 2002b. On limit points for some variants of rigid Lambek grammars. In P. Adriaans, H. Fernau, and M. van Zaanen, editors, *ICGI*, volume 2484 of *Lecture Notes in Artificial Intelligence*, pages 49–62. Springer-Verlag, September 23–25.

E. Mark Gold. 1967. Language identification in the limit. *Information and Control*, 10:447 474.

Makoto Kanazawa. 1998. *Learnable Classes of Categorial Grammars*. CSLI Publications, Stanford University, distributed by Cambridge University Press.

Maciej Kandulski. 1988. The equivalence of nonassociative Lambek categorial grammars and context-free grammars. *Zeischrift für Mathematische Logik und Grundlagne der Mathematik*, 34:41–52.

Shyam Kapur. 1991. *Computational Learning of Languages*. Available as technical report 91-1234, Department of Computer Science, Cornell University.

Natasha Kurtonina and Michael Moortgat. 1997. Structural control. In Patrick Blackburn and Maarten de Rijke, editors, *Specifying syntactic structures*, Studies in Logic, Language and Information. CSLI Publications, Stanford.

Joachim Lambek. 1958. The mathematics of sentence structure. *Amer. Math. Monthly*, 65:154–170.

Joachim Lambek. 1988. Categorial and categorical grammars. In Oehrle et al. (Oehrle et al., 1988), pages 297–317.

Michael Moortgat and Glyn Morrill. 1991. Heads and phrases. Type calculus for dependency and constituent structure. Manuscript.

Michael Moortgat. 1997. Categorial type logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 93–177. Elsevier Science B.V. Chapter 2.

Richard Moot. 2002. *Proof Nets for Linguistic Analysis*. Ph.D. thesis, Utrecht Institute of Linguistics OTS, Utrecht University.

R. T. Oehrle, E. Bach, and D. Wheeler, editors. 1988. *Categorial Grammars and Natural Language Structures*. Reidel, Dordrecht.

Mati Pentus. 1993. Lambek grammars are context free. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, pages 429–433, Los Alamitos, California. IEEE Computer Society Press.

Takeshi Shinohara. 1990a. Inductive inference from positive data is powerful. In *The 1990 Workshop on Computational Learning Theory*, pages 97–101, San Mateo, Calif. Morgan-Kaufmann.

Takeshi Shinohara. 1990b. Inductive inference of monotonic formal systems from positive data. In S. Arikawa, S. Goto, S. Ohsuga, and T. Yokomori, editors, *Algorithmic Learning Theory*, pages 339–351. Springer, New York and Berlin.

Hans-Jörg Tiede. 1998. Lambek calculus proofs and tree automata. In Michael Moortgat, editor, *Logical Aspects of Computational Linguistics Third International Conference, LACL'98, Selected Papers*, volume 2014 of *Lecture Notes in Artificial Intelligence*, pages 251–265, Grenoble, France, December. Springer-Verlag.

Hans-Jörg Tiede. 1999. *Deductive Systems and Grammars: Proofs as Grammatical Structures*. Ph.D. thesis, Illinois Wesleyan University.

Johan van Benthem. 1986. *Essays in Logical Semantics*. Reidel, Dordrecht.

Johan van Benthem. 1987. Categorial grammar and lambda calculus. In D. Skordev, editor, *Mathematical Logic and Its Applications*. Plenum Press, New York.

Johan van Benthem. 1991. *Language in Action: Categories, Lambdas and Dynamic Logic*, volume 130 of *Studies in Logic*. North-Holland, Amsterdam.