

# The Generative Capacity of the Lambek–Grishin Calculus: A New Lower Bound

Matthijs Melissen

Universiteit Utrecht, Postbus 80125, 3584 CS Utrecht,  
info@matthijsmelissen.nl,  
WWW home page: <http://www.matthijsmelissen.nl>

**Abstract.** The Lambek–Grishin calculus **LG** is a categorial grammar obtained by turning the one-sided sequents of the non-associative Lambek calculus **NL** into two-sided sequents, and adding interaction postulates between the two families of connectives thus obtained. In this paper, we prove a new lower bound on the generative capacity of **LG**, namely the class of languages that are the intersection of a context-free language and the permutation closure of a context-free language. This implies that **LG** recognizes languages like  $\{a^n b^n c^n d^n e^n \mid n \in \mathbb{N}\}$  and the permutation closure of  $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ .

## 1 Introduction

This paper studies the weak generative capacity, e.g. the class of languages that can be recognized, of the *Lambek–Grishin calculus* (**LG**) proposed by Moortgat [12]. We prove that any intersection of a context-free language and the permutation closure of a context-free language can be generated by **LG**.

The Lambek–Grishin calculus is an extension of the nonassociative Lambek calculus **NL** [10], obtained by allowing for structural multiplications on the right-hand side of sequents, and adding interaction principles between the families of connectives that correspond to the left-hand and right-hand multiplicative connectives. For **NL**, it has been proved by Kandulski [9] that this calculus is weakly equivalent to the context-free grammars. The same result for the associative variant of this calculus, the associative Lambek calculus **L**, has been proved by Pentus [15]. Context-free languages are however generally assumed to be insufficient for natural language parsing [6]. Moot [13] has proved that any *tree adjoining grammar* [7] can be converted into an **LG**-language, which implies that **LG** is at least *mildly context-sensitive*. Therefore **LG** seems to be more suitable for parsing natural language. The exact generative capacity of **LG** is currently unknown. Moot [13] also proposed a proof for conversion of **LG** into tree adjoining grammar, but this proof contained an error, as the current paper shows.

The language  $\{a^n b^n c^n d^n e^n \mid n \in \mathbb{N}\}$  cannot be generated by tree adjoining grammars (e.g. [8]), but we will show that this language can in fact be generated by **LG**. Note that neither the lower bound proved by Moot (the languages

generated by tree adjoining grammars) is a subset of the lower bound proved in this paper (the class of languages that are the intersection of a context-free language and the permutation closure of a context-free language), nor the other way around. This means that the current known lower bound on the generative capacity of **LG** is the union of those two classes. The only currently known upper bound on the generative capacity of **LG** is decidability [12], which makes the class of languages generated by this calculus at most recursive.

After the introduction, we first give a definition of the Lambek–Grishin calculus in Section 2. Then we proceed by proving the new lower bound on the generative complexity in Section 3. In Section 4 we give some examples of languages that can be generated, and in Section 5 some concluding remarks will be made.

## 2 The Lambek–Grishin calculus

The *Lambek–Grishin calculus* **LG** is an extension of the non-associative Lambek calculus **NL**. In **NL** there are only multiplicative connectives in a structural role on the left-hand side, while in **LG**, such connectives are also allowed on the right-hand side. We have therefore two different families of connectives, one related to each of both multiplications. **LG** is a deductive system. A sentence is grammatical if and only if the sequent belonging to this sentence can be derived in the deductive system.

**Definition 1.** *Sequents in **LG** have the form  $p \rightarrow q$  where  $p$  and  $q$  are types. We assign to every word in the language a finite number of types. We have a set of basic types called Atoms. The set of types is defined as follows: every atom is a type, and if  $p$  and  $q$  are types, then the following formulas are types as well:*

$$p \otimes q, \quad p \backslash q, \quad p / q, \quad p \oplus q, \quad p \odot q, \quad p \oslash q.$$

*An axiomatization of **LG** is displayed in Table 1.*

We can see  $\otimes$  as the connective playing a (multiplicative) structural role on the left-hand side of the arrow, and  $\oplus$  as the connective playing a (multiplicative) structural role on the right-hand side. Intuitively one or more words with type  $a \backslash b$  can be seen as a group of words which need a phrase of type  $a$  on their left, and return a phrase of type  $b$ . In the same way we can see a phrase of type  $a / b$  as a phrase which needs a phrase of type  $b$  on its right, before returning a  $b$  type phrase. Finally the  $\otimes$  operator can be seen as concatenation of words. The connectives  $\oplus$ ,  $\odot$  and  $\oslash$  are the duals of  $\otimes$ ,  $/$  and  $\backslash$  under arrow reversal, i.e. the symbol  $\oplus$  fulfills the same role on the right-hand side of the arrow as the symbol  $\otimes$  does on the left-hand side, and vice versa. Between the other connectives, similar relations hold.

There are multiple equivalent axiomatizations for **LG**, of which one is displayed in Table 1. Repetitions in derivations are not allowed. This axiomatization is particularly interesting because it is cut-free and allows for decidable proof search.

**Table 1.** Axioms and rules of Lambek–Grishin calculus.

$$\begin{array}{c}
\begin{array}{c}
t \rightarrow t \quad \mathbf{Id} \\
(t \in \mathit{Atoms})
\end{array}
\frac{b \rightarrow a \backslash c}{a \otimes b \rightarrow c} \mathbf{Res}
\quad
\frac{a \otimes c \rightarrow b}{c \rightarrow a \oplus b} \mathbf{Res} \\
\frac{a \rightarrow c/b}{c \circ b \rightarrow a} \mathbf{Res} \\
\frac{c \rightarrow a \quad b \rightarrow d}{a \backslash b \rightarrow c \backslash d} \mathbf{Mon}
\quad
\frac{d \rightarrow b \quad a \rightarrow c}{a/b \rightarrow c/d} \mathbf{Mon}
\quad
\frac{a \rightarrow c \quad b \rightarrow d}{a \otimes b \rightarrow c \otimes d} \mathbf{Mon} \\
\\
\frac{c \rightarrow a \quad b \rightarrow d}{a \otimes b \rightarrow c \otimes d} \mathbf{Mon}
\quad
\frac{d \rightarrow b \quad a \rightarrow c}{a \circ b \rightarrow c \circ d} \mathbf{Mon}
\quad
\frac{a \rightarrow c \quad b \rightarrow d}{a \oplus b \rightarrow c \oplus d} \mathbf{Mon} \\
\\
\frac{a \otimes (b \otimes c) \rightarrow d}{(a \otimes b) \otimes c \rightarrow d} \mathbf{G}_1
\quad
\frac{b \otimes (a \otimes c) \rightarrow d}{a \otimes (b \otimes c) \rightarrow d} \mathbf{G}_2
\quad
\frac{(a \otimes b) \circ c \rightarrow d}{a \otimes (b \circ c) \rightarrow d} \mathbf{G}_3
\quad
\frac{(a \otimes c) \circ b \rightarrow d}{(a \circ b) \otimes c \rightarrow d} \mathbf{G}_4
\end{array}$$

It uses only logical and no structural connectives; the role of the structural multiplications is played by  $\otimes$  on the left-hand side and  $\oplus$  on the right-hand side. We have *identity* (**Id**) as an axiom (where  $t$  is an atom). Furthermore there are two pairs of *residuation* (**Res**) rules. A double line denotes derivability in two directions. The left pair is the same as in **NL**, and the right pair is the symmetric version of the left pair. We also have six *monotonicity rules* (**Mon**). Finally the *Grishin interactions*  $\mathbf{G}_1 - \mathbf{G}_4$ , based on a paper by Grishin [5], are added. Those postulates govern the interaction between the  $\oplus$ - and  $\otimes$ -families.

**Lemma 1** ([12]). *Transitivity, i.e.  $a \rightarrow b$  and  $b \rightarrow c$  implies  $a \rightarrow c$  is admissible in **LG**.*

Now we will define how we can use **LG** to determine what sentences are grammatical.

**Definition 2.** *An **LG**-grammar has the form  $\mathcal{L}(\Sigma, s, \varphi)$ , where  $\Sigma$  is a finite alphabet,  $s$  the goal type (i.e. the type corresponding to an entire sentence) and  $\varphi$  a mapping called the type dictionary that assigns one or more types to every word. We require that for all  $t \in \Sigma$ ,  $\varphi(t)$  is finite. The language generated by an **LG**-grammar  $\mathcal{L}(\Sigma, s, \varphi)$  is defined as the set of all expressions  $\mathbf{t}_1 \dots \mathbf{t}_n$  over the alphabet  $\Sigma$  for which there exists a derivable sequent  $b_1 \otimes \dots \otimes b_n \rightarrow s$  (with some binary bracketing imposed on  $b_1 \otimes \dots \otimes b_n$ ) such that  $b_i \in \varphi(\mathbf{t}_i)$  for all  $1 \leq i \leq n$ .*

*Example 1.* Now we give an example derivation of the sentence ‘Alice thinks someone left’. The resulting formula on the top left is just an **NL**-derivable sequent. For clarity, the atoms occurring in the type of ‘someone’ are typeset in bold.

$$\begin{array}{c}
\frac{np \otimes (((np \setminus s) / s) \otimes (\mathbf{np} \otimes (np \setminus s))) \rightarrow \mathbf{s} \quad \mathbf{s} \rightarrow s}{(np \otimes (((np \setminus s) / s) \otimes (\mathbf{np} \otimes (np \setminus s)))) \circlearrowleft s \rightarrow \mathbf{s} \circlearrowleft \mathbf{s}} \text{Mon} \\
\frac{(np \otimes (((np \setminus s) / s) \otimes (\mathbf{np} \otimes (np \setminus s)))) \circlearrowleft s \rightarrow \mathbf{s} \circlearrowleft \mathbf{s}}{np \otimes (((np \setminus s) / s) \otimes (\mathbf{np} \otimes (np \setminus s))) \rightarrow (\mathbf{s} \circlearrowleft \mathbf{s}) \oplus s} \text{Res} \\
\frac{np \otimes (((np \setminus s) / s) \otimes (\mathbf{np} \otimes (np \setminus s))) \rightarrow (\mathbf{s} \circlearrowleft \mathbf{s}) \oplus s}{(\mathbf{s} \circlearrowleft \mathbf{s}) \circlearrowleft (np \otimes (((np \setminus s) / s) \otimes (\mathbf{np} \otimes (np \setminus s)))) \rightarrow s} \text{Res} \\
\frac{(\mathbf{s} \circlearrowleft \mathbf{s}) \circlearrowleft (np \otimes (((np \setminus s) / s) \otimes (\mathbf{np} \otimes (np \setminus s)))) \rightarrow s}{np \otimes ((\mathbf{s} \circlearrowleft \mathbf{s}) \circlearrowleft (((np \setminus s) / s) \otimes (\mathbf{np} \otimes (np \setminus s)))) \rightarrow s} \mathbf{G}_2 \\
\frac{np \otimes ((\mathbf{s} \circlearrowleft \mathbf{s}) \circlearrowleft (((np \setminus s) / s) \otimes (\mathbf{np} \otimes (np \setminus s)))) \rightarrow s}{np \otimes (((np \setminus s) / s) \otimes ((\mathbf{s} \circlearrowleft \mathbf{s}) \circlearrowleft (\mathbf{np} \otimes (np \setminus s)))) \rightarrow s} \mathbf{G}_2 \\
\frac{np \otimes (((np \setminus s) / s) \otimes ((\mathbf{s} \circlearrowleft \mathbf{s}) \circlearrowleft (\mathbf{np} \otimes (np \setminus s)))) \rightarrow s}{\underbrace{np}_{\text{Alice}} \otimes \underbrace{(((np \setminus s) / s)}_{\text{thinks}} \otimes \underbrace{((\mathbf{s} \circlearrowleft \mathbf{s}) \circlearrowleft \mathbf{np})}_{\text{someone}} \otimes \underbrace{(np \setminus s)}_{\text{left}})} \rightarrow s} \mathbf{G}_1
\end{array}$$

**Definition 3.** We will also use the product-free non-associative Lambek calculus or product-free **NL**, which we can obtain from the definition of **LG** by requiring that / and \ are the only connectives in the type dictionary.

Note that in product-free **NL**, we can leave out all rules containing the Grishin connectives  $\oplus$ ,  $\circlearrowleft$  and  $\circlearrowright$ .

**Definition 4.** We define a restricted categorial grammar called spinal **AB**-grammar ( $\mathbf{AB}_s$ ), where we only allow types of the form  $a$  and  $a \setminus b$ , where  $a$  and  $b$  are atomic types. The only rule is that we can replace any subexpression of the form  $a, a \setminus b$  by  $b$  in the left-hand side of the arrow.

It can easily be seen that all derivable sequents in  $\mathbf{AB}_s$  have the form  $(\dots (a_1, (a_1 \setminus a_2)), \dots), (a_{n-1} \setminus a_n) \rightarrow a_n$ . This calculus can be seen as a restricted version of the calculus used in Ajdukiewicz–Bar–Hillel grammar [1].

**Lemma 2.** Any  $\mathbf{AB}_s$ -sequent is an **LG**-sequent when we replace comma by  $\otimes$ .

*Proof.* Note that the rule of  $\mathbf{AB}_s$  can only be applied in positive context (i.e. in the antecedent of an even number of slashes) in  $\mathbf{AB}_s$ , namely in the antecedent of 0 slashes. We can easily check that the application of the rule of  $\mathbf{AB}_s$  in positive context is valid in **LG**.

**Definition 5.** The language generated by a product-free **NL**-grammar is defined in the same way as the language generated by an **LG**-grammar; the language generated by an  $\mathbf{AB}_s$ -grammar is also defined in the same way, except that we allow for an arbitrary number of goal types, instead of just one goal type.

**Definition 6.** We write  $a \div b$  as an expression which may stand for both  $a/b$  or  $b \setminus a$ . The antecedent of  $b \setminus a$ ,  $a/b$ ,  $b \circlearrowleft a$  and  $a \circlearrowright b$  is  $b$ . A subformula is positive resp. negative if it occurs in the antecedent of an even resp. odd number of slashes. The count of an atom in a type is the number of positive occurrences of this atom in the type minus the number of negative occurrences of this atom in the type, and the count of a set of atoms in a type is the sum of the count of those atoms.

**Lemma 3 (Count invariant, e.g. [17]).** For all sets of atoms  $A$ , if  $p \rightarrow q$  is derivable in **LG**, the count of  $A$  in  $p$  equals the count of  $A$  in  $q$ .

### 3 Main proof

**Theorem 1.** *Lambek–Grishin calculus is capable of recognizing any language that is the intersection of a context-free language and the permutation closure of a context-free language.*

The proof will follow at the end of this section. Because in **LG** we require non-empty left-hand sides in derivations, this calculus cannot generate the empty string, so we will ignore the recognition of the empty string in the equivalence proof.

**Definition 7.** *A finite state automaton (e.g. [16]) is a 5-tuple  $\langle Q, \Sigma, \delta, q_0, F \rangle$ , where  $Q$  is a finite set of states,  $\Sigma$  is a finite set of input symbols,  $\delta : Q \times \Sigma \rightarrow Q$  is the transition function,  $q_0 \in Q$  is the initial state and  $F \subseteq Q$  are the accept states. A string  $\mathbf{a}_1 \dots \mathbf{a}_n$  can be accepted by the finite state automaton if there is a sequence of states  $q_0, \dots, q_n$  such that  $q_n \in F$  and  $\delta(q_{i-1}, \mathbf{a}_i) = q_i$  for  $1 \leq i \leq n$ .*

*Example 2.* Consider the following finite state automaton:

$$\langle \{q_0, q_1, q_2, q_3, q_4\}, \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}, \delta, q_0, \{q_0, q_3\} \rangle$$

with  $\delta$  as follows:  $\delta(q_0, \mathbf{a}) = q_0$ ;  $\delta(q_0, \mathbf{b}) = q_1$ ;  $\delta(q_1, \mathbf{c}) = q_2$ ;  $\delta(q_2, \mathbf{d}) = q_3$ ;  $\delta(q_3, \mathbf{b}) = q_1$ ; for any other state  $q$  and input symbol  $i$ ,  $\delta(q, i) = q_4$ . This automaton recognizes an arbitrary number of  $\mathbf{a}$ 's, followed by an arbitrary number of sequences  $\mathbf{bcd}$ . Note that  $q_4$  acts here as a *sink*: it is not possible to leave this state.

**Lemma 4.** *For every regular language there is an equivalent  $\mathbf{AB}_s$ -language.*

*Proof.* It is well-known that the class of regular languages is exactly the class of languages recognized by finite state automata. We will show that any finite state automaton can be simulated by an  $\mathbf{AB}_s$ -grammar, thereby proving our lemma. Our proof is similar to the proof in [17], where it is proved that **LP**, the Lambek calculus with permutation, recognizes exactly all permutations of context-free languages. However, in that proof the definition of regular languages is used, while we make use of the machine model.

Given a finite state automata  $\langle Q, \Sigma, \delta, q_0, F \rangle$ , we define an  $\mathbf{AB}_s$ -grammar with symbols  $\Sigma'$ , goal types  $D$  and type dictionary  $\varphi$  as follows.

- $\Sigma' = \Sigma$ ;
- $D = F$ ;
- $\varphi$  is defined as follows:  $\varphi(\mathbf{a}) = \{q_1 \setminus q_2 \mid \delta(q_1, \mathbf{a}) = q_2\} \cup \{q \mid \delta(q_0, \mathbf{a}) = q\}$ .

In other words, if we can move with symbol  $\mathbf{a}$  from state  $q_1$  to  $q_2$ , we assign to  $\mathbf{a}$  the type  $q_1 \setminus q_2$ , and if we can immediately reach  $q$  from the initial state with symbol  $\mathbf{a}$ , we give a type  $q$ .

If a string  $\mathbf{t}_1 \dots \mathbf{t}_n$  of length at least 1 can be accepted by the finite state automaton, it means there is a sequence of states  $q_0, \dots, q_n$  such that  $q_n \in F$  and

$\delta(q_{i-1}, t_i) = q_i$  for  $1 \leq i \leq n$ . By our conversion, this means that  $q_n \in D$ ,  $q_1 \in \varphi(t_1)$  and  $q_i \setminus q_{i+1} \in \varphi(t_{i+1})$  for  $1 \leq i \leq n$ . Because  $q_1, q_1 \setminus q_2, \dots, q_{n-1} \setminus q_n \rightarrow q_n$  is derivable in  $\mathbf{AB}_s$ , the string  $t_1 \dots t_n$  can be recognized by the  $\mathbf{AB}_s$ -grammar.

If  $t_1 \dots t_n$  is recognized by an  $\mathbf{AB}_s$ -grammar, it means that there exists an  $\mathbf{AB}_s$ -derivable sequent  $b_1, \dots, b_n \rightarrow d$  such that  $b_i \in \varphi(t_i)$  for all  $1 \leq i \leq n$  and  $d \in D$ . Because all  $\mathbf{AB}_s$ -derivations have the form  $q_1, q_1 \setminus q_2, \dots, q_{n-1} \setminus q_n \rightarrow q_n$ , it holds that  $q_n = d$ ,  $q_1 \in \varphi(t_1)$  and  $q_{i-1} \setminus q_i \in \varphi(t_i)$  for  $2 \leq i \leq n$ . By definition of  $\varphi$ ,  $\delta(q_0, t_1) = q_1$  and  $\delta(q_{i-1}, t_i) = q_i$  for  $2 \leq i \leq n$ . By  $q_n = d$  and  $d \in D$ ,  $q_n \in D$  so also  $q_n \in F$ . Now we know that there is a sequence of states  $q_0, \dots, q_n$  such that  $q_n \in F$  and  $\delta(q_{i-1}, a_i) = q_i$  for  $1 \leq i \leq n$ , so  $t_1, \dots, t_n$  can be recognized by the finite state automaton.  $\square$

*Example 3.* Consider the finite state automaton from Example 2. We will create an  $\mathbf{AB}_s$ -language equivalent to this finite state automaton according to the procedure of Lemma 4. This gives us symbols  $\Sigma = \{a, b, c, d\}$ , goal types  $d = \{q_0, q_3\}$ , and the following type dictionary  $\varphi$ :

$p$	$\varphi(p)$
a	$q_0, q_0 \setminus q_0, q_1 \setminus q_4, q_2 \setminus q_4, q_3 \setminus q_4, q_4 \setminus q_4$
b	$q_1, q_0 \setminus q_1, q_1 \setminus q_4, q_2 \setminus q_4, q_3 \setminus q_1, q_4 \setminus q_4$
c	$q_4, q_0 \setminus q_4, q_1 \setminus q_2, q_2 \setminus q_4, q_3 \setminus q_4, q_4 \setminus q_4$
d	$q_4, q_0 \setminus q_4, q_1 \setminus q_4, q_2 \setminus q_3, q_3 \setminus q_4, q_4 \setminus q_4$

To show that the string  $aaabcd$  is in the language, it suffices to prove

$$q_0, q_0 \setminus q_0, q_0 \setminus q_0, q_0 \setminus q_1, q_1 \setminus q_2, q_2 \setminus q_3 \rightarrow q_3 \quad ,$$

which is valid in  $\mathbf{AB}_s$ . Of course, we could also simplify the type dictionary by leaving out the types with the sink state  $q_4$  in the right hand side of the  $\setminus$ .

**Lemma 5.** *For any product-free NL-language  $\mathcal{L}_1$  and  $\mathbf{AB}_s$ -language  $\mathcal{L}_2$  there is an LG-grammar that recognizes exactly the intersection of  $\mathcal{L}_1$  and the permutation closure of  $\mathcal{L}_2$ .*

*Proof.* Consider a language  $\mathcal{L}_1$  generated by a product-free NL-grammar  $\mathcal{G}_1$  with symbols  $\Sigma_1$ , goal type  $d$  and type dictionary  $\varphi_1$ , and a language  $\mathcal{L}_2$  generated by an  $\mathbf{AB}_s$ -grammar  $\mathcal{G}_2$  with symbols  $\Sigma_2$ , goal types  $D$  and type dictionary  $\varphi_2$ . Note that we can easily convert an  $\mathbf{AB}_s$ -grammar with multiple goal types  $D$  into an equivalent  $\mathbf{AB}_s$ -grammar with a single goal type, by choosing a fresh atomic goal type  $g$  and turning the type dictionary  $\varphi$  into  $\varphi'$ , where for every symbol  $t$ ,  $\varphi'(t) = \varphi(t) \cup \{a \setminus g \mid a \setminus d \in \varphi(t), d \in D\} \cup \{g \mid d \in \varphi(t), d \in D, d \text{ is atomic}\}$ . Therefore we can without loss of generality assume that  $D = \{d\}$  (i.e. we make sure that  $\mathcal{G}_1$  and  $\mathcal{G}_2$  have the same, single goal type), that  $d$  is atomic, that  $d$  does not occur in  $\varphi_2$  in the left-hand side of a  $\setminus$ , and that the intersection of the atoms in the range of  $\varphi_1$  and in the range of  $\varphi_2$  is  $\{d\}$ . We pick an atom  $s$  neither occurring in the range of  $\varphi_1$  nor in the range of  $\varphi_2$ ,  $s \neq d$ . We define  $T_1$  as the union of  $\{s\}$  and the atoms in the range of  $\varphi_1$  and  $T_2$  as the atoms in

the range of  $\varphi_2$ , both excluding  $d$ . Note that  $d$  functions as goal of the existing grammars  $\mathcal{G}_1$  and  $\mathcal{G}_2$ , while  $s$  will function as goal of the new grammar  $\mathcal{G}$ .

Now we can create an **LG**-grammar that recognizes the intersection of  $\mathcal{L}_1$  and the permutation closure of  $\mathcal{L}_2$  as follows. We take as symbols  $\Sigma_1 \cap \Sigma_2$ , and choose  $s$  as goal type. The type dictionary  $\varphi$  is defined in the following way. Given  $\mathbf{p} \in \Sigma$ , and  $\varphi_1(\mathbf{p})$  and  $\varphi_2(\mathbf{p})$ , we define  $\varphi(\mathbf{p})$  as follows:

$$\begin{aligned} \varphi(\mathbf{p}) = & \{(b \circ a) \circ c \mid c \in \varphi_1(p), a \setminus b \in \varphi_2(p)\} \\ & \cup \{(a \circ s) \circ c \mid c \in \varphi_1(p), a \in \varphi_2(p), a \text{ is atomic}\} . \end{aligned}$$

( $\Rightarrow$ ) First we will show that any string in the intersection of  $\mathcal{L}_1$  and the permutation closure of  $\mathcal{L}_2$  is also in  $\mathcal{L}$ . The steps we apply are displayed in Figure 1. When the bracketing of  $\otimes$  is not important, we just leave out the brackets. Assume we have a product-free **NL**-grammar  $\mathcal{G}_1$  and an **AB<sub>s</sub>**-grammar  $\mathcal{G}_2$ . We define  $\mathcal{G}$  to be the **LG**-grammar obtained by the procedure described above. Assume we have a string  $\mathbf{t}_1 \dots \mathbf{t}_n$  that is recognizable by  $\mathcal{G}_1$ , and a permutation  $\pi$  on  $[1..n]$  such that  $\mathbf{t}_{\pi(1)} \dots \mathbf{t}_{\pi(n)}$  is recognizable by  $\mathcal{G}_2$ . We will show that this permuted string is recognizable by  $\mathcal{G}$  as well. If  $\mathbf{t}_{\pi(1)} \dots \mathbf{t}_{\pi(n)}$  is recognizable by  $\mathcal{G}_2$ , there must be an **AB<sub>s</sub>**-derivation for (1) in Figure 1 (note that all **AB<sub>s</sub>**-derivable sequents have this shape) such that  $b_n = d$ ,  $b_1 \in \varphi_2(\mathbf{t}_{\pi(1)})$  and  $b_{i-1} \setminus b_i \in \varphi_2(\mathbf{t}_{\pi(i)})$  for every  $i$  with  $2 \leq i \leq n$ . By Lemma 2, (2) is an **LG**-derivable sequent. By symmetry under arrow reversal, (3) is also derivable. By residuation and monotonicity in **LG**, we can derive (4). Because  $\mathbf{t}_1 \dots \mathbf{t}_n$  is recognizable by  $\mathcal{G}_1$ , we have a product-free **NL**-derivation and therefore also an **LG**-derivation for  $a_1, a_2, \dots, a_n \rightarrow d$  such that  $a_i \in \varphi_1(\mathbf{t}_i)$  for every  $i$  with  $1 \leq i \leq n$ . Because  $b_n = d$ , we can obtain (5) by transitivity on this sequent and (4). Then we can move the  $b_i \circ b_{i-1}$ -subformulas to the left-hand side by residuation, obtaining (6). Finally we move those types in the right place with Grishin postulate **G<sub>1</sub>**. Note that we can place any  $b_i \circ b_{i-1}$  at any position under  $a_j$ , so we position them according to the permutation  $\pi$ , as depicted in (7), where  $\pi'$  is the inverse permutation of  $\pi$ . We now have a sequent of the form  $(\dots (c_1 \otimes c_2) \otimes \dots) \otimes c_n \rightarrow b_0$  where  $b_0$  is the goal type and each  $c_i$  has the form  $(b_{\pi'(i)} \circ b_{\pi'(i)-1}) \circ a_i$ . Because  $b_1 \in \varphi_2(\mathbf{t}_{\pi(1)})$ ,  $b_{i-1} \setminus b_i \in \varphi_2(\mathbf{t}_{\pi(i)})$  for  $2 \leq i \leq n$  and  $a_i \in \varphi_1(\mathbf{t}_i)$  for each  $1 \leq i \leq n$ , it holds that  $c_i \in \varphi(\mathbf{t}_i)$  for  $1 \leq i \leq n$  by definition of  $\varphi$ . Therefore  $\mathbf{t}_1 \dots \mathbf{t}_n$  is recognizable by  $\mathcal{G}$ . Now we can conclude that if a string can be recognized by a product-free **NL**-grammar and any permutation of it can be recognized by an **AB<sub>s</sub>**-grammar, the string can also be recognized by the corresponding **LG**-grammar obtained by the procedure described in this lemma.

( $\Leftarrow$ ) Now we show that any string in  $\mathcal{L}$  is both in  $\mathcal{L}_1$  and the permutation closure of  $\mathcal{L}_2$ . We prove this by showing reversibility of the steps in Figure 1. Assume that  $\mathbf{t}_1 \dots \mathbf{t}_n$  is in  $\mathcal{L}$ . Then there is a sequent  $d_1 \otimes \dots \otimes d_n \rightarrow d$  (with some binary bracketing imposed on the left-hand side) and  $d_i \in \varphi(\mathbf{t}_i)$  for every  $i$ . Sequents of this form will be called *initial*, which we will formalize in the next definition. Next, we will prove some properties about sequents of this form in the following lemmas, which we need to prove reversibility of (6)–(7). After that, we continue the current proof by proving reversibility of the other steps.

**Fig. 1.** The stepwise transformation of an  $\mathbf{AB}_s$ -sequent into an  $\mathbf{LG}$ -sequent. In every line, the difference with the next line is underlined.

$$(\dots (b_1, (b_1 \setminus b_2))_2 \dots)_2 (b_{n-1} \setminus b_n) \rightarrow b_n \quad (1)$$

$$\Downarrow$$

$$\underline{(\dots (b_1 \otimes (b_1 \setminus b_2)) \otimes \dots)} \otimes (b_{n-1} \setminus b_n) \rightarrow b_n \quad (2)$$

$$\Downarrow$$

$$b_n \rightarrow (b_n \otimes b_{n-1}) \oplus (\dots \oplus ((b_2 \otimes b_1) \oplus \underline{b_1}) \dots) \quad (3)$$

$$\Downarrow$$

$$\underline{b_n} \rightarrow (b_n \otimes b_{n-1}) \oplus (\dots \oplus ((b_2 \otimes b_1) \oplus ((b_1 \otimes b_0) \oplus b_0)) \dots) \quad (4)$$

$$\Downarrow$$

$$a_1 \otimes a_2 \otimes \dots \otimes a_n \rightarrow \underline{(b_n \otimes b_{n-1})} \oplus (\dots \oplus (\underline{(b_2 \otimes b_1)} \oplus (\underline{(b_1 \otimes b_0)} \oplus b_0)) \dots) \quad (5)$$

$$\Downarrow$$

$$\underline{(b_1 \otimes b_0)} \otimes (\underline{(b_2 \otimes b_1)} \otimes (\dots \otimes (\underline{(b_n \otimes b_{n-1})} \otimes (a_1 \otimes a_2 \otimes \dots \otimes a_n)) \dots)) \rightarrow b_0 \quad (6)$$

$$\Downarrow$$

$$((b_{\pi'(1)} \otimes b_{\pi'(1)-1}) \otimes a_1) \otimes ((b_{\pi'(2)} \otimes b_{\pi'(2)-1}) \otimes a_2) \otimes \dots \otimes ((b_{\pi'(n)-1} \otimes b_{\pi'(n)-1}) \otimes a_n) \rightarrow b_0 \quad (7)$$

**Definition 8.** Given two disjoint sets  $T_1, T_2$  and  $d \notin T_1 \cup T_2$ , we define initial sequents as sequents  $d_1, \dots, d_n \rightarrow d$  such that each  $d_i$  has the form  $(a_i \otimes b_i) \odot c_i$  with  $a_i \in T_2 \cup \{d\}$ ,  $b_i \in T_2$  and  $c_i \in T_1 \cup \{d\}$  (compare (7) in Figure 1).

In the following lemmas we will show that if an initial sequent can be derived, it can be done so by first moving all Grishin connectives outside the scope of any  $\otimes$ -connective (compare (6) in Figure 1). In other words, we show reversibility of the step (6)–(7) in this figure. This is the crucial part of our proof.

**Definition 9.** We define the following classes of types:

$$\begin{aligned}\mathcal{F} &::= c \mid \mathcal{F} \div \mathcal{F} \ ; \\ \mathcal{P} &::= (a \otimes b) \odot \mathcal{P} \mid \mathcal{F} \mid \mathcal{P} \otimes \mathcal{P} \ ; \\ \mathcal{Q} &::= \mathcal{F} \mid (a \otimes b) \oplus \mathcal{Q} \mid \mathcal{Q} \div \mathcal{P} \ ; \\ \mathcal{R} &::= a \mid (a \otimes b) \oplus \mathcal{R} \mid \mathcal{R} \div \mathcal{P} \ ;\end{aligned}$$

such that  $a \in T_2 \cup \{d\}$ ;  $b \in T_2$ ;  $c \in T_1 \cup \{d\}$ , where  $a, b$  and  $c$  are atomic types,  $T_1$  and  $T_2$  are disjoint classes of arbitrary types and  $d \notin T_1 \cup T_2$ .

Note that  $\mathcal{F}$  is the class of product-free Lambek types.

**Lemma 6.** Each initial sequent has the form  $p \rightarrow r$  with  $p \in \mathcal{P}$  and  $r \in \mathcal{R}$ .

*Proof.* This follows directly from Definition 8 (initial sequents).  $\square$

In other words, we can see sequents of the form  $p \rightarrow r$  with  $p \in \mathcal{P}$  and  $r \in \mathcal{R}$  as a generalization of the class of initial sequents. As will become clear, the former class can be seen as the closure of the class of initial sequents under operations such as Grishin interactions and various forms of residuation.

We also need to adapt the idea of ‘moving Grishin connectives outside the scope of any  $\otimes$ -connective’ to this generalization. We replace this notion with the notion of *internal* Grishin connectives. Intuitively, a Grishin connective is internal if it does not occur within the scope of a  $\otimes$ -connective, taking a restricted number of forms of residuation into account. Now we give a formal definition.

**Definition 10.** A subformula in a sequent  $p \rightarrow r$  with  $p \in \mathcal{P}$  and  $r \in \mathcal{R}$  is internal if (1) the type is in  $p$  while  $r$  contains  $\div$ ; (2) the type is in the left argument of  $\oplus$  in  $r$  while the right argument contains a  $\div$ -connective; (3) the type is within the scope of  $\otimes$  in  $p$ ; or (4) the type is in  $r$  in the right-hand argument of a  $\div$ -connective. Furthermore we say that a connective is internal if it occurs in an internal subformula.

**Lemma 7.** Consider the class of sequents  $p \rightarrow q$  such that  $p \in \mathcal{P}$  and  $q \in \mathcal{Q}$  and either  $p$  or  $q$  (or both) contain one of the Grishin connectives  $\otimes$ ,  $\odot$  and  $\oplus$ . Sequents in this class are underivable.

*Proof.* We prove that all derivations of sequents in this class would be infinite, by showing that to prove a sequent  $p \rightarrow q$  with  $p \in \mathcal{P}$  and  $q \in \mathcal{Q}$ , we first need to prove another sequent that is also in this class. Because derivations need to have a finite length, no sequents in this class are derivable. Remember that the decidable axiomatization consists of the identity axiom for atomic types, residuation, monotonicity and the Grishin interactions in rule-based form. We proceed by case distinction on each of the four rules that could have been used to derive  $p \rightarrow q$ .

First note that  $p \rightarrow q$  cannot be an axiom with atomic types, because the sequent is required to contain a connective.

Now we consider the case where the last proof step is monotonicity. Then the outer connectives on the left-hand and right-hand side must be equal. The only connectives that can appear as outer connective in both  $\mathcal{P}$  and  $\mathcal{Q}$  are  $/$  and  $\backslash$ . In that case,  $p$  does not contain Grishin connectives and therefore  $q$  does (by definition of  $\mathcal{Q}$ ), so we can write  $q$  as  $q' \div p'$  with  $p' \in \mathcal{P}$  and  $q' \in \mathcal{Q}$  and  $p$  as  $c_1 \div c_2$  with  $c_1, c_2 \in \mathcal{F}$ . Therefore the conclusion of the monotonicity rule has the form  $c_1 \div c_2 \rightarrow q' \div p'$ . Then we need to derive  $c_1 \rightarrow q'$  and  $p' \rightarrow c_2$ . Because  $q$  contains Grishin connectives, either  $p'$  or  $q'$  (or both) contains Grishin connectives. If  $p'$  contains Grishin connectives,  $p' \rightarrow c_2$  is again in the class and if  $q'$  contains Grishin connectives,  $c_1 \rightarrow q'$  is again in the class.

If the last step is residuation applied to  $p$ , then  $p$  must have the form  $(a \otimes b) \otimes p'$  or  $p_1 \otimes p_2$ . Then the premise is  $p' \rightarrow (a \otimes b) \oplus q$  in the first case and  $p_1 \rightarrow q/p_2$  or  $p_2 \rightarrow p_1 \backslash q$  in the second case, all of which are again in the class. Furthermore if the last step is residuation applied to  $q$ , then  $q$  must have one of the forms  $q' \div p'$ ,  $c_1 \div c_2$  or  $(a \otimes b) \oplus q'$  with  $a \in T_2 \cup \{d\}$ ,  $b \in T_2$ ,  $p' \in \mathcal{P}$ ,  $q' \in \mathcal{Q}$  and  $c_1, c_2 \in \mathcal{F}$ . If  $q = q' \div p'$  we have as premise  $p \otimes p' \rightarrow q'$  or  $p' \otimes p \rightarrow q'$ , both of which are again in the class. If  $q = c_1 \div c_2$  we have as premise  $p \otimes c_2 \rightarrow c_1$  or  $c_2 \otimes p \rightarrow c_1$ , which are again in the class as well. Finally if  $q = (a \otimes b) \oplus q'$ , the premise is either  $p \otimes q' \rightarrow a \otimes b$  or  $(a \otimes b) \otimes p \rightarrow q'$ , of which the latter is again in the class as well. The sequent  $p \otimes q' \rightarrow a \otimes b$  must have been derived by monotonicity, so one of the premises is  $b \rightarrow q'$ . We can easily check that the count of  $T_2$  in  $q'$  is less than or equal to 0, while the count of  $T_2$  in  $b$  is 1. Therefore  $b \rightarrow q'$  is undervivable by the count invariant, so we can conclude that  $p \rightarrow q$  cannot be derived.

Finally deriving  $p \rightarrow q$  with a Grishin postulate as last derivation step results in a premise which is again in the class.  $\square$

**Lemma 8.** *Consider the class of sequents  $p \rightarrow r$  that are obtained by substituting one or more internal  $\mathcal{P}$ -formulas that are positive in  $p_1$  or negative in  $r_1$  in  $p_1 \rightarrow r_1$  ( $p_1 \in \mathcal{P}$ ,  $r_1 \in \mathcal{R}$ ) by  $T_2$ -atoms. Sequents in this class are undervivable.*

*Proof.* Again we show that all derivations of sequents in this class would be infinite. We select a type introduced by the substitution and call it  $e$ . First note that because atomic sequents do not have internal subformulas, there are no axioms with atomic types in this class. Now we show that we can only prove a sequent  $p \rightarrow r$  from this class by proving another sequent that is also in this class. From this it follows that all derivations of sequents in this class would be

infinite. Now we proceed by case distinction on the possible proof steps. Just like in Lemma 7, there are two cases that do not lead to another sequent in this class straightforwardly.

The first nontrivial case is  $p \rightarrow (a \otimes b) \oplus r'$  with  $a \in T_2 \cup \{d\}$ ,  $b \in T_2$ ,  $p \in \mathcal{P}$  and  $r' \in \mathcal{R}$  preceded by right residuation, giving  $p \otimes r' \rightarrow a \otimes b$ , that must be preceded by monotonicity. Then the premises are  $p \rightarrow a$  and  $b \rightarrow r'$ . If  $e$  occurs in  $r$ , then  $e$  occurs in  $r'$  as well. Then  $e$  (and  $b$ ) are also internal in  $b \rightarrow r'$ , so this is in the class again. If not, then  $e$  must occur in  $p$ . By definition of *internal*, either a  $\div$  occurs in  $r$  and therefore also in  $r'$ , or  $e$  occurs within the scope of  $\otimes$ . In the first case,  $b$  in  $b \rightarrow r'$  is internal so this sequent is again in the class. In the second case,  $e$  is internal in  $p \rightarrow a$ , so now this sequent is in the class again.

The second case that cannot be handled straightforwardly is monotonicity on  $c_1 \div c_2 \rightarrow r' \div p'$  with  $c_1, c_2 \in \mathcal{F}$ ,  $p' \in \mathcal{P}$  and  $r' \in \mathcal{R}$ . Therefore the premises are  $c_1 \rightarrow r'$  and  $p' \rightarrow c_2$ . The types  $c_1$  and  $c_2$  cannot contain a  $\mathcal{P}$ -subtype, because they are themselves  $\mathcal{F}$ -subtypes, so  $e$  occurs in  $r' \div p'$ . If  $e$  occurs in  $r'$ , then  $c_1 \rightarrow r'$  is in the class again. Finally if  $e$  occurs in  $p'$ , then the count of  $T_2$  in  $p'$  is at least 1, while the count of  $T_2$  in  $c_2$  is 0, so  $p' \rightarrow c_2$  is underivable by the count invariant.  $\square$

**Lemma 9.** *If  $p \in \mathcal{P}$  and  $r \in \mathcal{R}$ , then if  $p \rightarrow r$  can be derived, there is a derivable sequent  $p_1 \rightarrow r_1$  with  $p_1 \in \mathcal{P}$  and  $r_1 \in \mathcal{R}$  such that  $p_1 \rightarrow r_1$  does not contain internal Grishin connectives and such that from  $p_1 \rightarrow r_1$  we can derive  $p \rightarrow r$  by just Grishin interactions and residuation, while every intermediate sequent in this derivation has the form  $p_2 \rightarrow r_2$  with  $p_2 \in \mathcal{P}$  and  $r_2 \in \mathcal{R}$ .*

Informally, this lemma tells us that every formula of the form  $p \rightarrow r$  with  $p \in \mathcal{P}$  and  $r \in \mathcal{R}$  can be derived with a proof in which the last proof steps consist of making the internal Grishin connectives in  $p \rightarrow r$  non-internal by Grishin interactions (and residuation). We can see this as a generalization of reversibility of steps (6)–(7) in Figure 1.

*Proof.* If  $p \rightarrow r$  does not contain internal Grishin connectives, we may set  $p_1 = p$  and  $r_1 = r$  so we are done. Now we assume  $p \rightarrow r$  contains internal Grishin connectives. We inspect the proofs of sequents  $p \rightarrow r$  with  $p \in \mathcal{P}$  and  $r \in \mathcal{R}$  (in the decidable axiomatization) and apply induction on the length of the proofs.

In the base case, where we consider all proofs of length 1, both  $p$  and  $r$  are atomic, so this sequent cannot contain internal Grishin connectives, which is a contradiction.

Now we continue with the induction step. In case the last proof step is monotonicity, the outer connective must be  $\div$ , so  $p = a_1 \div a_2$  with  $a_1, a_2 \in \mathcal{F}$  and  $r = r' \div p'$  with  $p' \in \mathcal{P}$  and  $r' \in \mathcal{R}$ , so the conclusion has the form  $a_1 \div a_2 \rightarrow r' \div p'$ . Then the premises are  $a_1 \rightarrow r'$  and  $p' \rightarrow a_2$ . In case  $p'$  contains Grishin connectives,  $p' \rightarrow a_2$  is underivable by Lemma 7, and in case  $p'$  does not contain Grishin connectives,  $r'$  must contain internal Grishin connectives, because  $r' \div p'$  must contain internal Grishin connectives. By induction hypothesis, if  $a_1 \rightarrow r'$  is derivable, it can be derived by making all internal Grishin connectives in  $r'$  non-internal by Grishin interactions and residuation. But then we could also

shift the application of the monotonicity rule up in the proof, by (from bottom to top) making the Grishin connectives in  $r'$  in  $a_1 \dot{\div} a_2 \rightarrow r' \dot{\div} p'$  non-internal by these steps (moving  $p'$  to the left by residuation, moving the Grishin connectives to the outside and moving  $p'$  back), and derive the premise thus obtained by the monotonicity rule.

If the last step in the proof is residuation applied to  $p$ , then  $p$  must have the form  $(a \otimes b) \otimes p'$  or  $p_1 \otimes p_2$ , so we have as premise  $p' \rightarrow (a \otimes b) \oplus r$  in the first case or  $p_1 \rightarrow r/p_2$  or  $p_2 \rightarrow p_1 \setminus r$  in the second case, on both of which the induction hypothesis applies. If the last step is residuation applied to  $r$ , then  $r$  has the form  $(a \otimes b) \oplus r'$  or  $r' \dot{\div} p'$ . In the latter case we need to prove that  $p \otimes p' \rightarrow r'$  or  $p' \otimes p \rightarrow r'$ , so the induction hypothesis applies. In the case  $r = (a \otimes b) \oplus r'$ , we need to show that either  $p \otimes r' \rightarrow a \otimes b$  or  $(a \otimes b) \otimes p \rightarrow r'$ . The induction hypothesis applies to the last sequent. The sequent  $p \otimes r' \rightarrow a \otimes b$  can only have been derived by monotonicity, so we need to derive  $p \rightarrow a$  and  $b \rightarrow r'$ . If  $r'$  contains  $\dot{\div}$ , then  $b$  is internal, so  $b \rightarrow r'$  is underivable by Lemma 8. If not, because  $p \rightarrow r$  contains an internal Grishin connective and  $r$  contains no  $\dot{\div}$ , clause (3) of the definition of *internal* applies. Therefore there is a Grishin connective within the scope of  $\otimes$  in  $p$ , so there is a Grishin connective in  $p \rightarrow a$ . Then by induction hypothesis  $p \rightarrow a$  is only derivable by first making all internal Grishin connectives in  $p$  noninternal. But then we can first make all internal Grishin connectives in  $p$  in  $p \rightarrow (a \otimes b) \oplus r'$  noninternal, before the monotonicity step.

Finally, if the last step is a Grishin interaction, we obtain a new sequent of the form  $p' \rightarrow r'$  with  $p' \in \mathcal{P}$  and  $r' \in \mathcal{R}$ , so the induction hypothesis applies.  $\square$

*Continuation of the proof of Lemma 5.* First note that  $d_1, \dots, d_n \rightarrow d$  is an initial sequent. We know by Lemma 6 that initial sequents have the form  $p \rightarrow r$  with  $p \in \mathcal{P}$  and  $r \in \mathcal{R}$ , and by Lemma 9 that if a sequent of this form can be derived, it can be done so by making all internal Grishin connectives non-internal by Grishin interactions (and residuation). Then we obtain a sequent of the form (6) in Figure 1, so (5) is also derivable (by residuation). The derivation of (5) consists of some steps that reduce the right-hand side to  $b_n$ , and some steps reducing the left-hand side (necessarily also to  $b_n$ ). The steps reducing the left-hand side to  $b_n$  permute with the steps reducing the right-hand side. If we reduce the right-hand side first, we obtain  $a_1 \otimes \dots \otimes a_n \rightarrow b_n$ , and therefore  $t_1 \dots t_n$  is recognizable by  $\mathcal{G}_1$ . If we reduce the left-hand side first, we obtain (4), showing that step (4)–(5) is reversible. Furthermore step (3)–(4) is reversible, because eliminating both  $b_0$ -atoms by monotonicity is the only way to derive (4), and the remaining residuation steps are reversible as well. Steps (1)–(3) are also reversible, so sequents of the form (1) are derivable, which shows that  $t_1 \dots t_n$  is in the permutation closure of  $\mathcal{G}_2$ .  $\square$

*Example 4.* Consider the  $\mathbf{AB}_s$ -language in Example 3. We translate this language into an  $\mathbf{LG}$ -language by the procedure of Lemma 5 such that the  $\mathbf{LG}$ -language recognizes the permutation closure of the  $\mathbf{AB}_s$ -language. We keep the symbols  $\Sigma$ , and choose a fresh goal type  $s$ . Furthermore we find the following

type dictionary  $\varphi'$ , with  $S \in \{r, r \setminus r, r \setminus q_0, r \setminus q_3\}$ .

$\mathbf{p}$	$\varphi'(\mathbf{p})$
<b>a</b>	$(q_0 \otimes s) \otimes S, (q_0 \otimes q_0) \otimes S, (q_4 \otimes q_1) \otimes S, (q_4 \otimes q_2) \otimes S, (q_4 \otimes q_3) \otimes S, (q_4 \otimes q_4) \otimes S$
<b>b</b>	$(q_1 \otimes s) \otimes S, (q_1 \otimes q_0) \otimes S, (q_4 \otimes q_1) \otimes S, (q_4 \otimes q_2) \otimes S, (q_1 \otimes q_3) \otimes S, (q_4 \otimes q_4) \otimes S$
<b>c</b>	$(q_4 \otimes s) \otimes S, (q_4 \otimes q_0) \otimes S, (q_2 \otimes q_1) \otimes S, (q_4 \otimes q_2) \otimes S, (q_4 \otimes q_3) \otimes S, (q_4 \otimes q_4) \otimes S$
<b>d</b>	$(q_4 \otimes s) \otimes S, (q_4 \otimes q_0) \otimes S, (q_4 \otimes q_1) \otimes S, (q_3 \otimes q_2) \otimes S, (q_4 \otimes q_3) \otimes S, (q_4 \otimes q_4) \otimes S$

With those types, we can parse any permutation of strings recognized by the finite state automaton in Example 2.

*Proof (Proof of Theorem 1).* As has been shown by Buszkowski [3], for every context-free language there is an equivalent product-free **NL**-grammar. Furthermore for every regular language we can find an equivalent **AB<sub>s</sub>**-language by Lemma 4. It has been proved [14] that the permutation closure of the regular languages is exactly the same as the permutation closure of the context-free languages. By Lemma 5, for any product-free **NL**-language  $\mathcal{L}_1$  and **AB<sub>s</sub>**-language  $\mathcal{L}_2$  there is an **LG**-grammar that recognizes exactly the intersection of  $\mathcal{L}_1$  and the permutation closure of  $\mathcal{L}_2$ . Therefore we can conclude that for any language that is the intersection of a context-free language and the permutation closure of a context-free language, we can find an equivalent **LG**-grammar.  $\square$

## 4 Examples

**Corollary 1.** *The language  $\{\mathbf{a}_1^n \dots \mathbf{a}_m^n \mid n \in \mathbb{N}\}$  can be recognized in **LG** for any  $m$ .*

Note that this class of languages cannot be generated by tree adjoining grammars (e.g. [8]). Therefore this example proves that **LG** generates more than tree adjoining grammars.

*Proof.* This language is the intersection of the permutation of the context-free language  $\{(\mathbf{a}_1 \dots \mathbf{a}_m)^n \mid n \in \mathbb{N}\}$  and the context-free language  $\{\mathbf{a}_1^* \dots \mathbf{a}_m^*\}$ .  $\square$

*Example 5.* The language  $\{\mathbf{a}^n \mathbf{b}^n \mathbf{c}^n \mathbf{d}^n \mathbf{e}^n \mid n \in \mathbb{N}\}$  can be recognized with the **LG**-grammar  $\langle \Sigma, s, \varphi \rangle$  with  $\Sigma = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}\}$ , goal type  $d$  and  $\varphi$  as follows, where  $a'_1 \in \{\mathbf{a}_1, d\}$  and  $r'_5 \in \{\mathbf{r}_5, d\}$ :

$\mathbf{p}$	$\varphi(\mathbf{p})$
<b>a</b>	$(a_2 \otimes a_1) \otimes r_1 \quad (a_2 \otimes a_1) \otimes (r_1 \setminus r_1)$ $(a_2 \otimes s) \otimes r_1 \quad (a_2 \otimes s) \otimes (r_1 \setminus r_1)$
<b>b</b>	$(a_3 \otimes a_2) \otimes (r_1 \setminus r_2) \quad (a_3 \otimes a_2) \otimes (r_2 \setminus r_2)$
<b>c</b>	$(a_4 \otimes a_3) \otimes (r_2 \setminus r_3) \quad (a_4 \otimes a_3) \otimes (r_3 \setminus r_3)$
<b>d</b>	$(a_5 \otimes a_4) \otimes (r_3 \setminus r_4) \quad (a_5 \otimes a_4) \otimes (r_4 \setminus r_4)$
<b>e</b>	$(a'_1 \otimes a_5) \otimes (r_4 \setminus r'_5) \quad (a'_1 \otimes a_5) \otimes (r_5 \setminus r'_5)$

**Corollary 2.** *A MIX-language, i.e. the permutation closure of a language consisting of strings of the form  $\mathbf{a}_1^n \dots \mathbf{a}_m^n$  with  $n \in \mathbb{N}$ , can be recognized in **LG** for any  $m$ .*

*Proof.* The language to be generated is the intersection of the permutation of the context-free language  $\{\mathbf{a}_1 \dots \mathbf{a}_m^n \mid n \in \mathbb{N}\}$  and the context-free language  $\{\mathbf{a}_k^* \mid 1 \leq k \leq m\}$ .  $\square$

*Example 6.* The permutation of the language  $\{\mathbf{a}^n \mathbf{b}^n \mathbf{c}^n \mid n \in \mathbb{N}\}$  can be recognized with the **LG**-grammar  $\langle \Sigma, s, \varphi \rangle$  with  $\Sigma = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ , goal type  $d$  and  $\varphi$  as follows, where  $a' \in \{a, d\}$  and  $r' \in \{r, d\}$ :

$$\frac{\mathbf{p} \mid \varphi(\mathbf{p})}{\begin{array}{l} \mathbf{a} \ (b \otimes a) \otimes r' \ (b \otimes a) \otimes (r \setminus r') \ (b \otimes s) \otimes r' \ (b \otimes s) \otimes (r \setminus r') \\ \mathbf{b} \ (c \otimes b) \otimes r' \ (c \otimes b) \otimes (r \setminus r') \\ \mathbf{c} \ (a' \otimes c) \otimes r' \ (a \otimes c) \otimes (r \setminus r') \end{array}}$$

## 5 Conclusion

We proved a new lower bound on the generative capacity of **LG**. This calculus recognizes all languages that are the intersection of a context-free language and the permutation closure of a context-free language. Therefore **LG** recognizes languages like the permutation closure of  $\{\mathbf{a}^n \mathbf{b}^n \mathbf{c}^n \mid n \in \mathbb{N}\}$  and for any  $m$  the language  $\{\mathbf{a}_1^n \dots \mathbf{a}_m^n \mid n \in \mathbb{N}\}$ . The latter language cannot be generated by tree adjoining grammars, which means that this paper proves a new lower bound on the generative complexity of the Lambek–Grishin calculus. In [13] it is claimed that **LG** is equivalent to the class of languages generated by the tree adjoining grammars, but as the current paper shows, this proof contains an error.

There is still a lot unknown about the generative capacity of **LG**. For example, it is not even known whether all recognizable languages are context-sensitive. Also it is neither obvious whether **LG** can be embedded in the indexed languages, nor the other way around. Another important unsolved question is the computational complexity of parsing. Both for cognitive plausibility and practical applicability a polynomial parsing algorithm would be desirable. There exist formalisms with a polynomial parsing algorithm that possibly have a similar generative complexity to **LG**, i.e. inclusion in either direction is open, such as Range Concatenation Grammars [2] and Global Index Grammars [4].

It is clear that the *strong* generative capacity of the (binary) Lambek–Grishin calculus is less than the one of tree adjoining grammars. This is partially because tree adjoining grammars allow for branching of arbitrary order, while binary Lambek–Grishin calculus forces us to impose a binary structure on languages to be parsed. This problem disappears when we use Lambek–Grishin calculus for  $n$ -ary connectives [11] instead of binary Lambek–Grishin calculus. It remains an open question whether the  $n$ -ary Lambek–Grishin calculus can indeed recognize all structures recognized by tree adjoining grammars.

## Acknowledgements

The author would like to thank Michael Moortgat and Vincent van Oostrom for their useful help and comments while supervising the MSc thesis of which this article is a result.

## References

1. Y. Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.
2. P. Boullier. Range concatenation grammars. In *New developments in parsing technology*, pages 269–289. Kluwer Academic Publishers, Norwell, MA, USA, 2004.
3. W. Buszkowski. Generative capacity of nonassociative lambek calculus. *Bulletin of Polish Academy of Sciences: Mathematics*, 34:507–516, 1986.
4. J. M. Castaño. Global index grammars and descriptive power. *J. of Logic, Lang. and Inf.*, 13(4):403–419, 2004.
5. V. N. Grishin. On a generalization of the ajdukiewicz-lambek system. In A.I. Mikhailov, editor, *Studies in Non-classical Logics and Formal Systems*, pages 315–343. Nauka, Moscow, 1983.
6. R. Huybregts. The weak inadequacy of context-free phrase structure grammars. In M. Trommelen G. J. de Haan and W. Zonneveld, editors, *Van Periferie Naar Kern*, pages 81–99. Foris Publications, Dordrecht, 1984.
7. Aravind Joshi, L.S. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163, 1975.
8. Aravind Joshi and Yves Schabes. Tree-adjoining grammars. In *Handbook of formal languages, vol. 3: beyond words*, pages 69–123. Springer-Verlag New York, Inc., New York, NY, USA, 1997.
9. M. Kandulski. The equivalence of nonassociative lambek categorial grammars and context-free grammars. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 34(1):41–52, 1988.
10. J. Lambek. On the calculus of syntactic types. In R. Jacobsen, editor, *Structure of Language and its Mathematical Aspects*, Proceedings of Symposia in Applied Mathematics, XII, pages 166–178. American Mathematical Society, 1961.
11. M. Melissen. Lambek–grishin calculus extended to connectives of arbitrary arity. In *Proceedings of the 20th Belgian-Netherlands Conference on Artificial Intelligence*, 2008.
12. M. Moortgat. Symmetries in natural language syntax and semantics: The lambek-grishin calculus. In D. Leivant and R. de Quieros, editors, *Proceedings WoLLIC '07*, pages 264–284. LNCS 4576. Springer, 2007.
13. R. Moot. Lambek grammars, tree adjoining grammars and hyperedge replacement grammars. In *Proceedings of the TAG+ Conference*. HAL - CCSD, 2008.
14. R. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.
15. M. Pentus. Lambek grammars are context free. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, pages 429–433, Los Alamitos, California, 1993. IEEE Computer Society Press.
16. M. Sipser. *Introduction to the Theory of Computation*. Course Technology, December 1996.
17. J. Van Benthem. *Language in action: categories, lambdes and dynamic logic*, volume 130 of *Studies in logic and the foundations of mathematics*. North-Holland, 1991.

The original publication is available at [www.springerlink.com](http://www.springerlink.com) via  
[http://dx.doi.org/10.1007/978-3-642-20169-1\\_8](http://dx.doi.org/10.1007/978-3-642-20169-1_8) .