On Translating Lambek Grammars with One Division into Context-Free Grammars

S. L. Kuznetsov*
Steklov Mathematical Institute, RAS
sk@mi.ras.ru

March 13, 2017

UDC 519.766.23

Abstract

In this paper we describe a method of translating a Lambek grammar with one division into an equivalent context-free grammar whose size is bounded by a polynom from the size of the original grammar. Earlier constructions by Buszkowski and Pentus lead to exponential growth of the grammar size.

1 Lambek Grammars and Context-Free Grammars

Lambek grammars were introduced by J. Lambek [12] for mathematical description of natural language fragments' syntax. They form a branch of the *categorial grammar* framework. Generally, grammars define *formal languages* (further we shall omit the word "formal"), *i.e.,* sets of *words* built from elements (*letters,* or *symbols*) of a finite set Σ called the *alphabet.* The set of all words over a given alphabet Σ is denoted by Σ^*; Σ^+ stands for the set

*This work is supported by the Russian Science Foundation under grant 14-50-00005.
This is a preprint of materials accepted for publication in Proceedings of the Steklov Institute of Mathematics.
Webpage of the Publisher of the original Russian version: http://www.maik.ru/

of all words except the empty one. The empty word is denoted by ϵ. We consider only grammars describing languages without the empty word (*i.e.,* subsets of Σ^+). However, the empty word can be used inside the grammar formalism.

A language can be infinite (as a set), and therefore the task of its description by means of a finite grammar is nontrivial (in particular, due to cardinality issues, not every language can be defined by a formal grammar of a particular kind).

A categorial grammar is a binary correspondence between letters of the alphabet and logical expressions called *syntactic types.* Each letter is associated with a finite number of syntactic types (but maybe more than one). A word[1] $w = a_1 \dots a_n$ belongs to the language defined by the grammar if and only if there exist syntactic types $A_1, \dots, A_n$, such that A_i is in the given correspondence with a_i (for all i from 1 to n) and the *sequent* $A_1 \dots A_n \to H$ is derivable in a specific logical calculus. Here H is a designated type (one for the whole grammar).

In Lambek grammars the calculus used to determine whether a word belongs to the language is the *Lambek calculus,* denoted by L. Syntactic types for the Lambek calculus are built from a set of *primitive types* (variables) Pr using three binary connectives, $\cdot$ (*multiplication*), $\backslash$, and $/$ (*left* and *right division*). The set of all types is denoted by Tp. We use capital Latin letters (A, B, C, ...) for types and capital Greek letters for finite (possibly empty) linearly ordered sequences of types. Following the linguistic tradition, we do not use commas to separate types in the sequence.

The Lambek calculus L derives objects of the form $\Pi \to B$ called *sequents.* Here B is a syntactic type and Π is a non-empty sequence of syntactic types.

The axioms of L are sequents of the form $A \to A$, and the rules of inference are as follows:

$$\frac{A\Pi \to B}{\Pi \to A \setminus B}\ \text{, where } \Pi \text{ is not empty} \qquad\qquad \frac{\Pi \to A \quad \Gamma B \Delta \to C}{\Gamma \Pi (A \setminus B) \Delta \to C}$$

[1]Here we should note some difference in terminology between the studies of "formal" and "real" languages. In linguistic applications *letters* of Σ correspond not to letters but rather to *words* (word forms) of the natural language; *words* over Σ correspond to *sentences.* Thus grammars describe not the lexical, but the syntactic level of language structure. In this paper we use the "letter–word" terminology, not the "word–sentence" one.

$$\frac{\Pi A \to B}{\Pi \to B \,/\, A}\text{ , where }\Pi\text{ is not empty} \qquad \frac{\Pi \to A \quad \Gamma B \Delta \to C}{\Gamma (B \,/\, A) \Pi \Delta \to C}$$

$$\frac{\Gamma \to A \quad \Delta \to B}{\Gamma \Delta \to A \cdot B} \qquad \frac{\Gamma A B \Delta \to C}{\Gamma (A \cdot B) \Delta \to C}$$

In addition to these rules, L admits the *cut rule* [12]:

$$\frac{\Pi \to A \quad \Gamma A \Delta \to C}{\Gamma \Pi \Delta \to C}$$

(This means that adding the cut rule does not lead to new derivable sequents.)

If $\Pi \to B$ is derivable in L, we denote this fact by $\mathrm{L} \vdash \Pi \to B$.

Finally, a Lambek grammar is a triple $\mathcal{G} = \langle \Sigma, \rhd, H \rangle$, where Σ is an alphabet, $H \in \mathrm{Tp}$ is a designated type, and $\rhd \subset \Sigma \times \mathrm{Tp}$ is a finite binary correspondence between letters of the alphabet and syntactic types. As said before, a word $w = a_1 \dots a_n$ is accepted by $\mathcal{G}$ iff there exist types $A_1, \dots, A_n$ such that $a_i \rhd A_i$ (for all i from 1 to n) and $\mathrm{L} \vdash A_1 \dots A_n \to H$.

Note that Lambek grammars as defined above are explicitly disallowed to generate the empty word. However, one can drop the constraint of Π being non-empty from the rules, which leads a modified calculus L^*. Grammars based on L^* can define languages containing the empty word.

Another way of describing language syntax, introduced by Chomsky [4], is the framework of *context-free grammars.* A context-free grammar is a quadruple $G = \langle N, \Sigma, P, S \rangle$, where Σ is the alphabet which we define the language over, N is an auxiliary alphabet called the set of *nonterminal symbols* (N and Σ are required to be disjoint), $S \in N$ is the *start symbol*, and P is a finite set of *rules* of the form $A \Rightarrow \alpha$, where $A \in N$ and α is a word in the alphabet $(N \cup \Sigma)$. In our definition α in every rule is required to be non-empty. The symbol A and the word α are called the *left-hand side* and the *right-hand side* of the rule respectively. Let η and θ be arbitrary (possibly empty) words over $(N \cup \Sigma)$. Then if $(A \to \alpha) \in P$, the word $\eta\alpha\theta$ is *directly derivable* from the word $\eta A \theta$ in G (notation: $\eta A \theta \Rightarrow_G \eta\alpha\theta$). The relation $\Rightarrow_G^*$ ("derivable in G") is defined as the reflexive-transitive closure of $\Rightarrow_G$. The *language defined by the grammar* G is the set $\{w \in \Sigma^+ \mid S \Rightarrow_G^* w\}$. Such languages are called *context-free* (recall that we consider only languages without the empty word).

If G is a grammar (it could be a Lambek grammar or a context-free one), then the language defined by G will be denoted by $\mathcal{L}(G)$.

1. *Each language defined by a Lambek grammar is context-free. Each context-free language is defined by a Lambek grammar.*

The first statement of this theorem was proved by Pentus [17]; for the case where only one division operation is used, this result was proved earlier by Buszkowski [3]. The second statement was proved by Gaifman [2] for a weaker formalism calles basic categorial grammars, or Ajdukiewicz–Bar-Hillel grammars; Buszkowski [3] noticed that Gaifman's construction also works for Lambek grammars. In this construction only one division operation ($\backslash$, or, symmetrically, $/$) is used.

Theorem 1 states equivalence of Lambek grammars and context-free grammars *in the weak sense.* That is, the classes of languages as sets of words, defined by these two grammar formalisms, coincide. Actually, both Lambek grammars and context-free ones can solve a more sophisticated task than just determining whether a word belongs to the language. For a word belonging to the language, they can assign an extra structure encoding the *semantics* ("meaning") of the given word. If this extra structure is also preserved in the transformation, the grammars are *equivalent in the strong sense* [9][10]. In this paper we consider only equivalence in the weak sense.

Besides the Lambek calculus L itself, its extensions and fragments are also broadly studied. Many extensions of the Lambek calculus are important for linguistic applications [8][13][7][14]. Connectives of the Lambek calculus have a natural interpretation as operations (multiplication and two divisions) on formal languages [16], therefore it looks a very natural idea to extend the Lambek calculus with other well-known language-theoretic operations. An interesting open problem here is connected with the iteration ("Kleene star"). The question of extending the Lambek calculus with iteration looks easy, yet no "good" axiomatisation is known; however, there exist complete systems with an ω-rule or infinite branches in derivation trees [11]. Interestingly enough, in analogous systems for Gödel–Löb modal logic GL and its extensions infinite derivations can be "wrapped" into finite (cyclic) ones [26][27]. Possibly, a variant of this strategy could also be successful for the Lambek calculus with iteration.

For the fragments of L, the cut elimination theorem makes their axiomatisation an easy task: one just leaves only the rules operating the chosen connectives.

In this paper we consider the fragment of the Lambek calculus with only one division, L($\backslash$). As opposed to the full Lambek calculus (L) and its fragments with two operations (L($\backslash, \cdot$), L($/, \cdot$), L($/, \backslash$)) with NP-complete derivability problems [18][25][24], the derivability problem for L($\backslash$) is decidable in polynomial time [23]. On the other hand, any context-free language (see above) can be defined by an L($\backslash$)-grammar. In other words, only one Lambek operation, namely one of two divisions, is sufficient for modelling context-free derivations.

2 Savateev's Derivability Criterion for L($\backslash$)

In this section we formulate a graph-theoretic criterion for derivability in L($\backslash$), proved by Savateev [22]. We shall use this criterion in our construction.

Let $\mathrm{Atn} = \{p^{(i)} \mid p \in \mathrm{Pr}, i \in \mathbb{N}\}$ be the set of *atoms*. (An atom is a primitive type with a numerical superscript.) We are going to consider finite non-empty sequences of atoms; the set of all such sequences is denoted by Atn^+.

For $\mathbb{A} = p_1^{(i_1)} p_2^{(i_2)} \dots p_k^{(i_k)} \in \mathrm{Atn}^+$ let $\mathbb{A}^{+2} = p_1^{(i_1+2)} p_2^{(i_2+2)} \dots p_k^{(i_k+2)}$.

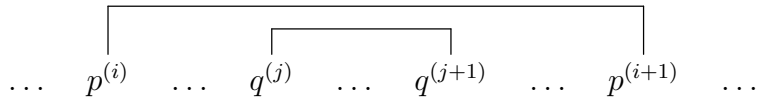
Define two mappings $\gamma, \bar{\gamma} \colon \mathrm{Tp} \to \mathrm{Atn}^+$ of L($\backslash$) types to sequences of atoms:

$$\begin{array}{ll} \gamma(p) = p^{(1)} & \bar{\gamma}(p) = p^{(2)} \\ \gamma(A \setminus B) = \bar{\gamma}(A)\gamma(B) & \bar{\gamma}(A \setminus B) = \bar{\gamma}(B)\big(\gamma(A)\big)^{+2} \end{array}$$

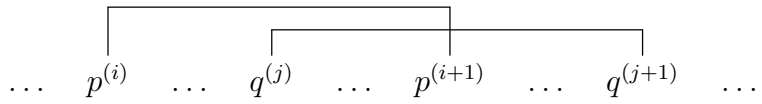
Let $\mathbb{A}$ be a finite sequence of atoms. A *proof net* on $\mathbb{A}$ is a pairing on $\mathbb{A}$ such that:

1. every element belongs exactly to one pair;
2. each pair consist of $p^{(i)}$ and $p^{(i+1)}$, where $p \in \mathrm{Pr}$ and $i \in \mathbb{N}$, and $p^{(i)}$ lies to the left from $p^{(i+1)}$ in the sequence;
3. one can draw the pairing as lines in the upper semiplane without intersections; in other words, two pairs are allowed two be located like this:

or like this:



but not like this:



4. if the left atom in a pair has an even superscript, then between the atoms of the pair there exists an atom with the superscript less than the superscripts of both elements of this pair.

Note that in his another paper [23] Savateev uses a slightly different criterion: for a pair in which the left atom has an even superscript 2ℓ there should be an atom with superscript *precisely* $2\ell-1$ in between. This criterion is equivalent to the one described above.

2. *A sequent* $A_1, \ldots, A_n \to B$ *is derivable in* $\mathrm{L}(\backslash)$ *if and only if there exists a proof net on* $\gamma(A_1) \ldots \gamma(A_n)\bar{\gamma}(B)$. [22]

3 The Growth of the Lambek Grammar Size when Translating into a Context-Free Grammar

By Theorem 1, between the two grammar formalisms (Lambek grammars and context-free grammars) there exist translations in both directions that support weak equivalence of the formalisms. In such situations, however, (see, for example, [21]) one usually asks not only about the existence of such translations, but also about the *change* (usually growth) *of the size* of the object (in our case, the grammar) after such transformation. If the size grows dramatically (*e.g.,* exponentially), the translation becomes practically useless.

Unfortunately, the translation of Lambek grammars into context-free ones, presented by Pentus [17], leads to an exponential growth of the grammar size. In the general case such inefficiency appears to be inevitable, because

the derivability problem for L is NP-complete (see above), while the parsing problem for context-free grammars is decidable in polynomial time.

For the one-division case another method for translating L(\)-grammars into context-free grammars was proposed earlier by Buszkowski [3], but it also leads to exponential growth of the grammar size. Here we present a method that translates an L(\)-grammar into a context-free grammar of the size bounded by a polynom of the original grammar size.

First we define the notion of the *size* for Lambek grammars and context-free grammars more accurately.

Let $A \in \mathrm{Tp}$. Define the size of A as the number of primitive type occurrences in A; denote it by $|A|$. The formal definition of $|A|$ is recursive: $|p_i| = 1$; $|A \setminus B| = |B / A| = |A \cdot B| = |A| + |B|$. The size of a Lambek grammar $G = \langle \Sigma, \rhd, H \rangle$ is defined as follows: $|G| = |H| + \sum\limits_{\langle a, A \rangle \in \rhd} |A|$.

Let us call the size of a context-free grammar $G = \langle N, \Sigma, P, S \rangle$ the number $\sum\limits_{(A \to \alpha) \in P} |\alpha|$, where $|\alpha|$ is the number of letters in α. Note that $|\Sigma| \leq |G|$ (we assume that Σ does not contain useless symbols not appearing in $\mathcal{L}(G)$), $|N| \leq |G|$, and $|P| \leq |G|$.

The size of any grammar G is denoted by $|G|$ (this applies both to Lambek grammars and to context-free ones).

3. *For any* L(\)*-grammar there exists an equivalent (in the weak sense) context-free grammar whose size is bounded by a polynom of the size of the original grammar.*

4 Savateev's Conditions as Context-Free Rules

The main idea of our translation of grammars is to write the proof net conditions as a context-free grammar. Unfortunately, the sequences of atoms used in the definition of proof net are words over an infinite alphabet (Atn), therefore it is formally impossible to construct a context-free grammar for a set of them. To overcome this issue, we restrict Atn to a finite set.

Let G be an L(\)-grammar. First we remove all primitive types not occurring in G, from Pr. After that Pr becomes a finite set. Moreover, it has at most $|G|$ elements.

Now we define the *depth* of a syntactic type from Tp(\) recursively: $d(p_i) = 1$; $d(A \setminus B) = \max\{d(A) + 1, d(B)\}$. On the other hand, denote

the accordingly restricted set of atoms, $\{p^{(i)} \mid p \in \mathrm{Pr}, i \leq m\}$, by Atn_m ($m \in \mathbb{N}$).

1. *If* $A \in \mathrm{Tp}(\backslash)$, *then* $\gamma(A) \in \mathrm{Atn}^+_{d(A)}$ *and* $\bar{\gamma}(A) \in \mathrm{Atn}^+_{d(A)+1}$.

Proof. We proceed by structural induction on A. If $A = p_i$, then $d(A) = 1$, $\gamma(A) = p_i^{(1)} \in \mathrm{Atn}^+_1$, and $\bar{\gamma}(A) = p_i^{(2)} \in \mathrm{Atn}^+_2$.

For the $A = B \setminus C$ case we first notice some obvious properties of the sets Atn^+_m. First, if $\mathbb{B} \in \mathrm{Atn}^+_{m_1}$ and $\mathbb{C} \in \mathrm{Atn}^+_{m_2}$, then $\mathbb{B}\mathbb{C} \in \mathrm{Atn}^+_{\max\{m_1,m_2\}}$. Second, if $\mathbb{B} \in \mathrm{Atn}^+_m$, then $\mathbb{B}^{+2} \in \mathrm{Atn}^+_{m+2}$.

Finally,

$$\gamma(B \setminus C) = \bar{\gamma}(B)\gamma(C) \in \mathrm{Atn}^+_{\max\{d(B)+1,d(C)\}} = \mathrm{Atn}^+_{d(B \setminus C)};$$

$$\bar{\gamma}(B \setminus C) = \bar{\gamma}(C)\big(\gamma(B)\big)^{+2} \in \mathrm{Atn}^+_{\max\{d(C)+1,d(B)+2\}} = \mathrm{Atn}^+_{d(B \setminus C)+1}.$$

□

We call the *depth* of a L(\)-grammar $G = \langle \Sigma, \rhd, H \rangle$ the number $d(G) = \max\{d(A) \mid a \rhd A \quad a \in \Sigma \quad A = H\}$. Note that since $d(A) \leq |A|$ for any type A (this is easily checked by induction), for any L(\)-grammar G we have $d(G) \leq |G|$. [2]

Let $m = d(G)+1$. Then for any sequent used for checking whether a word belongs to the language defined by G the corresponding sequence of atoms lies in Atn^+_m. On the other hand, $|\mathrm{Atn}_m| = |\mathrm{Pr}| \cdot m \leq |G|(d(G)+1) \leq |G|(|G|+1)$, therefore the sequences of atoms to be considered are words in an alphabet of a polynomially bounded (w.r.t. the size of the original grammar) cardinality.

Now let us define the context-free grammar $\mathbf{S}_m$ that formalises the existence of a proof net on a given word over the alphabet Atn_m. Let $\mathbf{S}_m = \langle N, \mathrm{Atn}_m, P, S \rangle$, where $N = \{S, R_1, \ldots, R_{m-1}\}$ and P consists of the following rules:

[2] At the same time, if G contains a "very deep" complicated syntactic type, $d(G)$ could be close to $|G|$.

$S \Rightarrow R_k$	for every k from 1 to $m-1$;
$R_{k_1} \Rightarrow R_{k_2}$,	if $k_1 > k_2$;
$R_k \Rightarrow p^{(2\ell-1)} R_k p^{(2\ell)}$,	if $k < m$ and $2\ell < m$;
$R_k \Rightarrow p^{(2\ell)} R_k p^{(2\ell+1)}$,	if $2\ell + 1 < m$ and $k < 2\ell$;
$R_{2\ell-1} \Rightarrow p^{(2\ell-1)} S p^{(2\ell)}$,	if $2\ell < m$;
$R_{2\ell-1} \Rightarrow p^{(2\ell-1)} p^{(2\ell)}$,	if $2\ell < m$;
$R_k \Rightarrow R_k S$	for every k from 1 to $m-1$;
$R_k \Rightarrow S R_k$	for every k from 1 to $m-1$.

The total number of rules is not greater than $m + m^2 + 2 \cdot |\mathrm{Pr}| \cdot m^2 + 2 \cdot |\mathrm{Pr}| \cdot m + 2m \leq 8 \cdot |\mathrm{Pr}| \cdot m^2 \leq 8|G|^3$. The right-hand side of each rule has length not greater than 3. Thus, $|\mathbf{S}_m| \leq 24|G|^3$, *i.e.,* the size of $\mathbf{S}_m$ is bounded by a polynom of the size of the original grammar G.

2. *A sequence of atoms $\mathbb{A} \in \mathrm{Atn}_m^+$ has a proof net if and only if it belongs to the language described by $\mathbf{S}_m$.*

Proof. In order to use induction, we augment the statement of this lemma ($S \Rightarrow_{\mathbf{S}_m}^* \mathbb{A}$ if and only if $\mathbb{A}$ has a proof net) with analogous statements about other nonterminal symbols: $R_k \Rightarrow_{\mathbf{S}_m}^* \mathbb{B}$ if and only if $\mathbb{B}$ has a proof net and contains an atom with superscript not greater than k.

Now these statements are easily proved by simultaneous induction: in the "if" part the induction parameter is the derivation length in $\mathbf{S}_m$, for the "only if" part we proceed by induction on the length of the sequence of atoms. □

Now let us describe a method to obtain a context-free grammar, weakly equivalent to G, from $\mathbf{S}_m$. We'll need the notion of *homomorphism* of formal languages.

Let Σ_1 and Σ_2 be two alphabets. A homomorphism is a mapping $h\colon \Sigma_1^* \to \Sigma_2^*$ such that $h(uv) = h(u)h(v)$ for any $u, v \in \Sigma_1^*$. Clearly, one can arbitrarily define h on elements of Σ_1, and then it is uniquely propagated to longer words from Σ_1^*; $h(\epsilon)$ is always ϵ (otherwise the equality $h(a) = h(a\epsilon) = h(a)h(\epsilon)$ gets violated).

An important particular case of homomorphism is *non-extending homomorphism,* that maps every letter $a \in \Sigma_1$ either to the empty word or to one letter from Σ_2 (but not to a longer word).

Further we shall denote the alphabet Atn_m by Σ_1.

Recall that we are constructing a context-free grammar for the language defined by the L(\)-grammar $G = \langle \Sigma, \rhd, H \rangle$. Let us introduce an auxiliary

alphabet $\Sigma_2 = \{\langle a, A\rangle \mid a \rhd A\}$ and a fresh symbol \$ that doesn't belong to alphabets introduced before. Define two homomorphisms $g\colon \Sigma_2 \cup \{\$\} \to \Sigma_1$ $h\colon \Sigma_2 \cup \{\$\} \to \Sigma$ as follows:

$$\begin{array}{ll} g(\langle a, A\rangle) = \gamma(A); & h(\langle a, A\rangle) = a; \\ g(\$) = \bar{\gamma}(H); & h(\$) = \epsilon. \end{array}$$

It's easy to see that if $\mathbf{S}_m$ defines the language M, then the language defined by G is equal to $h\big(g^{-1}(M) \cap \{u\$ \mid u \in \Sigma_2^+\}\big)$. One can construct a context-free grammar for this language, because the operations of taking the preimage of a homomorphism, intersecting with a regular language, and application a homomorphism preserve context-freeness of the language. These facts were independently noticed by several authors [5][15][28] and nowadays are included into standard textbooks in formal language theory [1][6][19]. However, we'll have to carefully analyse the proofs in order to get an estimation of the size of the yielded context-free grammar.

5 Estimation of the Size of the Context-Free Grammar

For the "inverse homomorphism" construction (the step from M to $g^{-1}(M)$) we shall use an auxiliary notion of *pushdown automaton (PDA).* A PDA is an automaton equipped with a pushdown memory structure (or *stack*). We are going to use the following version of the PDA definition: a PDA is a sextuple $\mathfrak{M} = \langle Q, \Sigma, \Gamma, \Delta, q_0, Z_0\rangle$, where Q, Σ, and Γ are finite set, q_0 $in Q$, $Z_0 \in \Gamma$, and Δ is a finite subset of the Cartesian product $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times Q \times \Gamma^*$. The set A is called the set of *states,* q_0 is the *initial state,* Z_0 is the *starting stack symbol.* We shall write elements of Δ (*transitions* of the automaton) of the form $\langle p, a, Z, q, \beta\rangle$ in the following way: $p \xrightarrow{a,\ Z:\beta} q$.

A *configuration* of a PDA is a triple $\langle q, v, \gamma\rangle$, where $q \in Q$, $v \in \Sigma^*$, $\gamma \in \Gamma^*$. Informally this means that the automaton is now in the state q with the word v not yet read from the input (v is a suffix of the word originally given to the automaton), and the stack contains γ.

The initial configuration of the pushdown automaton has the form $\langle q_0, w, Z_0\rangle$, where w is the word on which we run the automaton.

Applying a transition $p \xrightarrow{a,\ Z:\beta} q$ changes the configuration as follows: $\langle p, aw, Z\gamma\rangle \to_{\mathfrak{M}} \langle q, w, \beta\gamma\rangle$. Informally, if the automaton was in state p with

Z on top of the stack[3], the automaton reads a from the input (or reads nothing if $a = \epsilon$), removes Z from the top of the stack, puts the word β onto the stack, and changes the state to q. Note that this definition requires the automaton to pop exactly one symbol from the stack (of course, it can always immediately put it back by adding it to β). As usually, $\to^*_{\mathfrak{M}}$ is the reflexive-transitive closure of $\to_{\mathfrak{M}}$.

Finally, a word w is *accepted* by a PDA $\mathfrak{M}$, or, in other words, belongs to the *language defined by* $\mathfrak{M}$, if $\langle q_0, w, Z_0\rangle \to^*_{\mathfrak{M}} \langle q, \epsilon, \epsilon\rangle$ for some[4] state $q \in Q$.

Note that $\mathfrak{M}$, in general, operates *nondeterministically, i.e.,* it could be possible to apply several different transitions from one configuration. A word is accepted by the automaton, if at least one sequence of transitions succeeds (*i.e.,* doesn't terminate in the middle of the input word and ends by a configuration with nothing on the stack).

The further plan of building a context-free grammar for $\mathcal{L}(G)$ is as follows. First we transform the grammar $\mathbf{S}_m$ to a PDA that defines the same language $M = \mathcal{L}(\mathbf{S}_m)$. Next we build a PDA for $g^{-1}(M) \cap \{u\$ \mid u \in \Sigma_2^+\}$. After that we return to context-free grammars (transform this PDA into a context-free grammar). Finally, by applying the non-extending homomorphism h, we obtain a context-free grammar for $\mathcal{L}(G) = h\big(g^{-1}(M) \cap \{u\$ \mid u \in \Sigma_2^+\}\big)$.

Constructions of automata and grammars used in our transformations were taken from [1] and [6]. In these books one can find correctness proofs (*i.e.,* proofs of the facts that these automata and grammars define the needed languages). Here we focus on estimating the size of the resulting context-free grammar.

For estimating the complexity (size) of a PDA $\mathfrak{M} = \langle Q, \Sigma, \Gamma, \Delta, q_0, Z_0\rangle$ we shall use the following parameters: first, the number of transitions ($|\Delta| = \delta(\mathfrak{M})$); second, the number $d(\mathfrak{M}) = \max\big\{|\beta| \mid \big(p \xrightarrow{a,\ Z:\beta} q\big) \in \Delta\big\}$, that characterises the maximal number of letters put onto the stack at one transition; third, the number of states ($|Q| = q(\mathfrak{M})$).

3. *There exists a PDA $\mathfrak{M}_1$ that defines the language $M = \mathcal{L}(\mathbf{S}_m)$* ([1], Lemma 2.24). *Moreover, $\delta(\mathfrak{M}_1) \le 2|\mathbf{S}_m|$, $q(\mathfrak{M}_1) = 1$, and $d(\mathfrak{M}_1) = 3$.*

Proof. Let N be the set of all non-terminal symbols of $\mathbf{S}_m$.

[3]In our formalism the stack "grows" to the left.

[4]As one can notice, in this definition the halting condition for PDA is emptiness of the stack rather than moving to a special "halting" state.

The automaton $\mathfrak{M}_1$ contains only one state q_0 and is equal to $\langle\{q_0\}, \Sigma_1, N \cup \Sigma_1, \Delta_1, q_0, S\rangle$. Here we take the union of the main and auxiliary alphabet of $\mathbf{S}_m$ as the stack alphabet, and the starting stack symbol is the start symbol of the grammar.

The set of transitions Δ_1 is built in the following way:

1. for each rule $A \Rightarrow \alpha$ of $\mathbf{S}_m$ we add a transition $q_0 \xrightarrow{\epsilon,\ A:\alpha} q_0$;
2. for each $a \in \Sigma_1$ we add a transition $q_0 \xrightarrow{a,\ a:\epsilon} q_0$.

The number of transitions of the first type is equal to the number of rules in $\mathbf{S}_m$. The number of transitions of second type is equal to $|\Sigma_1|$. Both numbers are less or equal to $|\mathbf{S}_m|$. Therefore, $\delta(\mathfrak{M}_1) \leq 2|\mathbf{S}_m|$.

The equality $d(\mathfrak{M}_1) = 3$ is due to the fact that right-hand sides of all rules in $\mathbf{S}_m$ contain at most three symbols. □

4. *Let $\mathfrak{M}_1$ be a PDA for M as constructed in the previous lemma, and let $g\colon \Sigma_2 \cup \{\$\} \to \Sigma_1$ be a homomorphism such that $|g(a)| \leq n$ for every $a \in \Sigma_2$. Then there exists a PDA $\mathfrak{M}_2$ that defines the language $g^{-1}(M) \cap \{u\$ \mid \Sigma_2^+\}$* ([6], Theorem 7.30, and [1], Lemma 2.22 and Theorem 2.26)*. Moreover, $q(\mathfrak{M}_2) \leq 2(n+1) \cdot |\Sigma_2| + 2$, $\delta(\mathfrak{M}_2) \leq (2n+6) \cdot |\mathbf{S}_m| \cdot |\Sigma_2| + 2|\mathbf{S}_m| + 2|\Sigma_2| + 2$, and $d(\mathfrak{M}_2) = 3$.*

Proof. Let us build the PDA $\mathfrak{M}_2 = \langle Q_2, \Sigma_2 \cup \{\$\}, \Gamma_2, \Delta_2, q_0, S\rangle$ in the following way. Let $\Gamma_2 = N \cup \Sigma_1 \cup \{\#\}$. The new symbol # will play the role of *stack bottom marker,* forbidding the automaton to halt too early, even if the stack is empty and the input word is read up to the end.

For Q_2 we take the set of all pairs of the form $\langle i, x\rangle$, where $i \in \{0, 1\}$ and $x \in \Sigma_1^*$ is a suffix of a word $g(a)$ for some letter $a \in \Sigma_2$ (in particular, x can be empty; states $\langle 0, \epsilon\rangle$ and $\langle 1, \epsilon\rangle$ are going to play a special role in $\mathfrak{M}_2$), and two additional states q_0 (the initial one) and q_F (the final one).

Finally, the transitions of $\mathfrak{M}_2$ (elements of Δ_2) are as follows:

1. $q_0 \xrightarrow{\epsilon,\ S:S\#} \langle 0, x\rangle$;
2. $\langle 0, \epsilon\rangle \xrightarrow{a,\ X:X} \langle 0, g(a)\rangle$ for every $X \in \Gamma_2$ and $a \in \Sigma_2$;
3. $\langle 0, \epsilon\rangle \xrightarrow{a,\ X:X} \langle 1, g(a)\rangle$ for every $X \in \Gamma_2$ and $a \in \Sigma_2$;

4. $\langle 1, \epsilon\rangle \xrightarrow{\$,\ X:X} \langle 1, g(\$)\rangle$ for every $X \in \Gamma_2$;

5. $\langle i, bv\rangle \xrightarrow{\epsilon,\ X:\alpha} \langle i, v\rangle$, if $q_0 \xrightarrow{b,\ X:\alpha} q_0$ is a transition of $\mathfrak{M}_1$;

6. $\langle 1, \epsilon\rangle \xrightarrow{\epsilon,\ \#:\epsilon} q_F$.

The workflow of $\mathfrak{M}_2$ can be informally interpreted as follows. The first transition puts a special symbol # to the bottom of the stack. Since the only way to remove it is the sixth transition, the automaton could finish his work only in the q_F state. Next, by the second transition, the automaton reads the ongoing symbol a and stores the word $g(a)$ in the "internal memory" (the second component of the state). Several applications of the fifth transition emulate the work of $\mathfrak{M}_1$ on this word. After this emulation the "internal memory" is again empty, and $\mathfrak{M}_2$ is ready for reading the next letter of the input. The next-to-the-last letter is read by the third transition, changing the first component of the state from 0 to 1. This guarantees that the word is non-empty and doesn't consist of only one symbol \$. After that we again apply the fifth transition several times. Finally, the fourth transition and several applications of the fifth one do the same job for \$. This ensures that this symbol appears only once and is located at the end of the word (*i.e.,* this handles the intersection with $\{u\$ \mid u \in \Sigma_2^+\}$).

Correctness of this construction easily follows from the proofs of the propositions from [1] and [6] mentioned in the statement of the lemma.

It's easy to see that $d(\mathfrak{M}_2) = d(\mathfrak{M}_1) = 3$. Let us estimate $\delta(\mathfrak{M}_2) = |\Delta_2|$. We have $|\Gamma_2| \cdot |\Sigma_2|$ transitions of type 2 and the same number of transitions of type 3. The number of transitions of type 4 is equal to $|\Gamma_2|$. Since the length of each word $g(a)$ ($a \in \Sigma_2$) is not greater than n, every such word has not more than $(n+1)$ suffixes (from the empty word to the whole one). The total number of such suffixes is less or equal to $(n+1) \cdot |\Sigma_2|$. Each transition of type 5 is defined by such a suffix and a transition of $\mathfrak{M}_1$, whence the number of such transitions is not greater than $(n+1) \cdot |\Sigma_2| \cdot \delta(\mathfrak{M}_1)$. Finally, transitions of types 1 and 6 are unique, so there are 2 of them.

Thus, the total number of transitions in $\mathfrak{M}_2$ is not greater than $2|\Gamma_2| \cdot |\Sigma_2| + |\Gamma_2| + (n+1) \cdot |\Sigma_2| \cdot \delta(\mathfrak{M}_1) + 2$. Recalling that $|\Gamma_2| = |N| + |\Sigma_1| + 1 \le 2|\mathbf{S}_m| + 1$ and $\delta(\mathfrak{M}_1) \le 2|\mathbf{S}_m|$, we get the required estimation $(2n+6) \cdot |\mathbf{S}_m| \cdot |\Sigma_2| + 2|\mathbf{S}_m| + 2|\Sigma_2| + 2$.

Finally, the estimation $q(\mathfrak{M}_2) = |Q_2| \le 2(n+1) \cdot |\Sigma_2| + 2$ also follows from the estimation of the number of suffixes of words of the form $g(a)$. □

5. *Let* $\mathfrak{M}_2 = \langle Q_2, \Sigma_2 \cup \{\$\}, \Gamma_2, \Delta_2, q_0, S\rangle$ *be a PDA that defines a language over alphabet* $\Sigma_2 \cup \{\$\}$. *Then there exists a context-free grammar* G_2 *defining the same language* ([1], Lemma 2.26). *Moreover,* $|G_2| \le (d(\mathfrak{M}_2)+1)\cdot\delta(\mathfrak{M}_2)\cdot \big(q(\mathfrak{M}_2)\big)^{d(\mathfrak{M}_2)} + q(\mathfrak{M}_2)$.

Proof. The desired grammar $G_2 = \langle N_2, \Sigma_2, P_2, S_2\rangle$ is built as follows.

For non-terminal symbols of this grammar we use triples $\langle q, Z, r\rangle$, where $q, r \in Q_2$ and $Z \in \Gamma_2$, and an also a special symbol S_2. Following [1], we denote $\langle q, Z, r\rangle$ by $[qZr]$ (note that is *one* symbol of N_2).

The rules of G_2 (elements of P_2) are the following ones:

1. $[qZr] \Rightarrow a[rX_1s_1][s_1X_2s_2]\dots[s_{k-1}X_ks_k]$,
   if $q \xrightarrow{a,\ Z:X_1\dots X_k} r$ is a transition of $\mathfrak{M}_2$; $a \in \Sigma_2 \cup \{\$, \epsilon\}$, and s_1, …, s_k are arbitrary elements of Q_2;
   in particular, for a transition of the form $q \xrightarrow{a,\ Z:\epsilon} r$ we add the rule $[qZr] \Rightarrow a$;
2. $S_2 \Rightarrow [q_0Sq]$ for every $q \in Q_2$.

For each rule of the first type, $k \le d(\mathfrak{M}_2)$. Since every such rule corresponds to a transition of $\mathfrak{M}_2$ and k states $s_1, \dots, s_k \in Q_2$, the total number of such rules is not greater than $\delta(\mathfrak{M}_2)\cdot\big(q(\mathfrak{M}_2)\big)^{d(\mathfrak{M}_2)}$, and the total length of right-hand sides of these rules is less or equal to $(d(\mathfrak{M}_2)+1)\cdot\delta(\mathfrak{M}_2)\cdot\big(q(\mathfrak{M}_2)\big)^{d(\mathfrak{M}_2)}$.

The number of rules of the second type is equal to $|Q_2| = q(\mathfrak{M}_2)$, and the right-hand side of every such rule contains only one letter. This yields the required inequality for $|G_2|$. □

Let us simplify the estimation of $|G_2|$. By construction, $|\Sigma_2| \le |G|$, and for every $a \in \Sigma_2$ we have $|g(a)| \le |G|$, where $|G|$ is the original L(\)-grammar. Moreover, $|\mathbf{S}_m| \le 24|G|^3$. Let $n = |G|$. By Lemma 4, $d(\mathfrak{M}_2) = 3$, $q(\mathfrak{M}_2) \le 2(n+1)\cdot n + 2 = 2n^2 + 2n + 2$ and $\delta(\mathfrak{M}_2) \le (2n+6)\cdot 24n^3\cdot n + 2\cdot 24n^3 + 2n + 2 = 48n^5 + 144n^4 + 48n^3 + 2n + 2$.

Therefore, $|G_2| \le 4\cdot(48n^5 + 144n^4 + 48n^3 + 2n + 2)\cdot(2n^2 + 2n + 2)^3 + 2n^2 + 2n + 2 = O(n^{11})$.

Finally, applying a non-extending homomorphism (h) to right-hand sides of G_2 doesn't increase its size and gives a context-free grammar of size $O(|G|^{11})$ for the language $h(g^{-1}(M) \cap \{u\$ \mid u \in \Sigma_2^+\} = \mathcal{L}(G)$. This finishes the proof of Theorem 3.

6 Conclusion

Despite the fact that the full Lambek calculus L is NP-complete, and therefore Theorem 3 could hardly be generalised to all Lambek grammars, for grammars where all types have constantly bounded depth there exists a polynomial ($O(n^4)$) algorithm for checking whether a word belongs to the language defined by such a grammar [20]. Therefore the question of translating of a Lambek grammar with types of bounded depth into a context-free grammar of polynomial size is interesting for further investigation.

The polynomial ($O(n^3)$) algorithm for checking derivability in L($\backslash$) [23] is, essentially, the application of the standard dynamic programming technique (the Cocke–Younger–Kasami algorithm, CYK) to the grammar $\mathbf{S}_m$. Unfortunately, this grammar is ambiguous (for some words it yields more that one parsing tree), whence most of the efficient parsing algorithms are not applicable to this grammar of work slower than CYK. Moreover, the size of $\mathbf{S}_m$ depends on the complexity of the original sequent, therefore even the universal Valiant's algorithm [29] on this grammar works slower than CYK. Nevertheless, the presented interpretation of Savateev's criterion in terms of context-free rules could be useful in attempts of constructing a more efficient algorithm for checking derivability in L($\backslash$).

Acknowledgments

The author is grateful to Prof. G. Morrill (Barcelona) and Prof. M.R. Pentus (Moscow) for fruitful discussions.

References

[1] *Aho A.V., Ullman J.D.* The theory of parsing, translation, and compiling. Vol. 1: Parsing. Englewood Cliffs, NJ: Prentice-Hall, 1972.

[2] *Bar-Hillel Y., Gaifman C., Shamir E.* On the categorial and phrase-structure grammars. Bull. Res. Council Israel, Section F. 1960. Vol. 9F. P. 1–16.

[3] *Buszkowski W.* The equivalence of unidirectional Lambek categorial grammars and context-free grammars // Z. math. Logik Grundl. Math. 1985. Bd. 31. S. 369–384.

[4] *Chomsky N.* Three models for the description of language // IRE Trans. Inform. Theory. 1956. Vol. 2. P. 113–124.

[5] *Evey J.* Application of pushdown store machines // Proc. Fall Joint Computer Conference. Montvale, NJ: AFIPS Press, 1963. P. 215–227.

[6] *Hopcroft J.E., Motwani R., Ullman J.D.* Introduction to automata theory, languages and computation. 2nd edition. Addison-Wesley, 2001.

[7] *Jäger G.* On the generative capacity of multi-modal categorial grammars // Research Lang. Comput. 2003. V. 1. No. 1–2. P. 105–125.

[8] *Kanazawa M.* The Lambek calculus enriched with additional connectives // J. Log. Lang. Inform. 1992. V. 1. P. 141–171.

[9] *Kanazawa M., Salvati S.* The string-meaning relations definable by Lambek grammars and context-free grammars // Formal Grammar: Proc. 17th and 18th Intl. Confs., FG 2012/2013 / Ed. by G. Morrill, M.-J. Nederhof. Berlin: Springer, 2013. P. 191–208. (Lect. Notes Comput. Sci.; V. 8036).

[10] *Kuznetsov S.L.* On Translating Context-Free Grammars into Lambek Grammars // Proc. Steklov Inst. Math. 2015. Vol. 290. P. 63–69.

[11] *Kuznetsov S.L., Ryzhkova N.S.* A fragment of the Lambek calculus with iteration (in Russian) // Mal'tsev Meeting 2015. Collection of Abstracts. Novosibirsk, 2015.

[12] *Lambek J.* The mathematics of sentence structure // Amer. Math. Month. 1958. V. 65, No. 3. P. 154–170.

[13] *Moortgat M.* Multimodal linguistic inference // J. Log. Lang. Inform. 1996. V. 5. No. 3–4. P. 349–385.

[14] *Morrill G.* Categorial grammar. Logical syntax, semantics, and processing. Oxford Univ. Press, 2011.

[15] *Oettinger A.G.* Automatic syntactic analysis and pushdown store // Proc. Symp. Appl. Math. 1961. V. 12. AMS, Providence, RI.

[16] *Pentus M.* Models for the Lambek calculus // Ann. Pure Appl. Log. 1995. Vol. 75. No. 1–2. P. 179–213.

[17] *Pentus M.* Lambek calculus and formal grammars // Provability. Complexity. Grammars. AMS Transl. Ser. 2. Vol. 192. Providence, RI: AMS, 1999.

[18] *Pentus M.* Lambek calculus is NP-complete // Theor. Comput. Sci. 2006. V. 357. P. 186–201.

[19] *Pentus A.E., Pentus M.R.* Mathematical theory of formal languages (textbook, in Russian). Moscow: INTUIT / BINOM Publishers, 2009.

[20] *Pentus M.* A polynomial-time algorithm for Lambek grammars of bounded order // Linguistic Analysis. 2010. V. 36. No. 1–4. P. 441–471.

[21] *Podolskii V.V.* Circuit complexity meets ontology-based data access // Proc. 10th Intl. Comput. Sci. Symp. in Russia, CSR 2015. Berlin: Springer, 2015. P. 7–26. (Lect. Notes Comput. Sci.; V. 9139).

[22] *Savateev Yu.* Lambek grammars with one division are decidable in polynomial time // Computer Science — Theory and Applications / Editors E.A. Hirsch et al. Berlin: Springer, 2008. P. 273–282. (Lect. Notes Comput. Sci., V. 5010).

[23] *Savateev Yu.V.* Recogintion of derivability for the Lambek calculus with one division // Moscow Univ. Math. Bull. 2009. Vol. 64. No. 2. P. 73–75.

[24] *Savateev Yu.V.* Algorithmic complexity of fragments of the Lambek calculus: PhD Thesis (in Russian). Moscow State University, 2009.

[25] *Savateev Yu.* Product-free Lambek calculus is NP-complete // Ann. Pure Appl. Log. 2012. V. 163. Iss. 7. P. 775–788.

[26] *Shamkanov D.S.* Circular proofs for the Gödel–Löb provability logic // Math. Notes. 2014. Vol. 96. No. 4. P. 575–585.

[27] *Shamkanov D.* Nested sequents for provability logic GLP // Log. J. IGPL. 2015. V. 3. No. 5. P. 789–815.

[28] *Schützenberger M.-P.* On context-free languages and pushdown automata // Information and Control. 1963. V. 6. No. 3. P. 246–264.

[29] *Valiant L.G.* General context-free recognition in less than cubic time // J. Comp. Syst. Sci. 1975. V. 10. P. 308–315.