Non-associative Lambek Categorial Grammar in Polynomial Time

Erik Aarts[1]

Research Institute for Language and Speech
Utrecht University
Trans 10
3512 JK Utrecht

Dep. of Mathematics and Computer Science
University of Amsterdam
Plantage Muidergracht 24
1018 TV Amsterdam

Kees Trautwein

Research Institute for Language and Speech
Utrecht University
Trans 10
3512 JK Utrecht

Abstract

We present a new axiomatization of the non-associative Lambek categorial grammar. We prove that it takes polynomial time to translate any non-associative Lambek categorial grammar into an equivalent context-free grammar. Since it is possible to recognize a sentence generated by a context-free grammar in polynomial time, this proves that a sentence generated by any non-associative Lambek categorial grammar can be recognized in polynomial time.

1 Introduction

In this paper we present an algorithm for the recognition of sentences generated by non-associative Lambek categorial grammars (see Lambek (1958) for details on the Lambek categorial grammar). First we will define categorial grammars. These grammars consist of a lexicon defining a type assignment to words, and a calculus defining the well-formed sequences of types. An important categorial grammar is the Lambek grammar which uses the Lambek calculus to define well-formedness. The complexity of the recognition problem for Lambek categorial grammar is still unknown. So far only nondeterministic polynomial time algorithms are known. On the other hand there isn't any proof that this recognition problem is NP-complete.

For the second order fragment polynomial algorithms have been found (Aarts, 1993; Aarts, to appear). Another interesting fragment of the Lambek calculus is the

[1]The author was sponsored by project NF 102/62-356 ('Structural and Semantic Parallels in Natural Languages and Programming Languages'), funded by the Netherlands Organization for the Advancement of Research (NWO).

non-associative calculus. The non-associative Lambek calculus can be obtained from the standard Lambek calculus by dropping the associativity rule. There are several ways to axiomatize the resulting calculus. Buszkowski (1986) proves that the non-associative Lambek categorial grammar is equivalent with context-free grammar. The grammar that he obtains has exponential size. Only exponential time algorithms have been found in the past (Janssen, 1991; Trautwein, 1991). In this paper we present a new axiomatization of the non-associative Lambek calculus. This axiomatization enables us to give a polynomial translation into context-free grammars. After the translation we can use any context-free recognition algorithm to recognize a sentence generated by the non-associative Lambek categorial grammar in polynomial time.

2 Categorial Grammars

A categorial grammar G is defined by a lexicon Lex and a calculus C. A (finite) set of *primitive types* Pr is given. Tp is the set of *types*. Types are constructed from primitive types by two type-forming operators: / and \, i.e., Tp is the smallest set including Pr such that if $x, y \in Tp$, then $x\backslash y, x/y \in Tp$. One member s of Pr is singled out as the *distinguished type*.

A *lexicon* Lex is a finite relation between an alphabet Σ and Tp ($Lex \subset \Sigma \times Tp$, $\Sigma \cap Tp = \emptyset$). If a lexicon Lex relates $a \in \Sigma$ with $A \in$ Tp ($\langle a, A\rangle \in Lex$), we say Lex *assigns A to a*.

STp is the set of *strings of types*. Furthermore, we define the set $BSTp$, of *bracketed strings of types* as the smallest set including Tp such that if $X, Y \in BSTp$, then $(X, Y) \in BSTp$. The *yield* of a bracketed string X is the string we get when we erase the brackets. Yields are elements of STp.

Now $L(G)$ is defined to be the set of strings $w \in \Sigma^*$, $w = a_1 \dots a_n$ such that for some bracketed string of types X, $yield(X) = A_1, \dots, A_n$, $\langle a_i, A_i\rangle \in Lex$ $(1 \leq i \leq n)$ and $X \rightarrow s$ is derivable in the calculus C.

We use the non-associative Lambek calculus as the calculus C. There is a number of possible axiomatizations for this calculus. We choose one given in Kandulski (1988) as our starting point. This calculus is called NLG_0 and can be found in Figure 1.

$$[\mathrm{A1}] \quad x \rightarrow x \text{ , for } x \in Tp$$

$$\frac{(X, y) \rightarrow x}{X \rightarrow x/y}\,[\mathrm{R1}'] \qquad\qquad \frac{(y, X) \rightarrow x}{X \rightarrow y\backslash x}\,[\mathrm{R1}'']$$

$$\frac{X \rightarrow y \quad Y[x] \rightarrow z}{Y[(x/y, X)] \rightarrow z}\,[\mathrm{R2}'] \qquad \frac{X \rightarrow y \quad Y[x] \rightarrow z}{Y[(X, y\backslash x)] \rightarrow z}\,[\mathrm{R2}'']$$

for $X, Y \in BSTp$ and $x, y, z \in Tp$.

Figure 1: NLG_0

The notation $Y[x]$ (resp. $Y[X]$) is used to indicate the bracketed string of types Y, in which, on a certain place, type x (resp. bracketed string of types X) occurs.

Before we show that sentences can be recognized in polynomial time we first introduce two auxiliary calculi: NLG_0^* and NLG_0^{**}. The calculus NLG_0^* differs from NLG_0 in that the X's in the R1$'$ and R1$''$ rules are restricted: they must be in Tp instead of in $BSTp$. The calculus NLG_0^{**} differs from NLG_0^* in that the X's in the R2$'$ and R2$''$ rules must be in Tp instead of in $BSTp$.

The calculus NLG_0^{**} can be written down as follows (compared to NLG_0, X has been replaced by w):

$$[\mathrm{A1}]\quad x \to x \text{ , for } x \in Tp$$

$$\frac{(w,y) \to x}{w \to x/y}\,[\mathrm{R1}'] \qquad \frac{(y,w) \to x}{w \to y\backslash x}\,[\mathrm{R1}'']$$

$$\frac{w \to y \quad Y[x] \to z}{Y[(x/y,w)] \to z}\,[\mathrm{R2}'] \qquad \frac{w \to y \quad Y[x] \to z}{Y[(w,y\backslash x)] \to z}\,[\mathrm{R2}'']$$

$$\text{for } Y \in BSTp \text{ and } w,x,y,z \in Tp.$$

Figure 2: NLG_0^{**}

In the sequel, we are going to prove that NLG_0, NLG_0^* and NLG_0^{**} are equivalent. We prove the following inclusions:

- $\mathrm{NLG}_0 \subseteq \mathrm{NLG}_0^*$ (Theorem 2.1)
- $\mathrm{NLG}_0^* \subseteq \mathrm{NLG}_0^{**}$ (Theorem 2.2)
- $\mathrm{NLG}_0^{**} \subseteq \mathrm{NLG}_0$ (Theorem 2.3)

Theorem 2.1 *For all k, if some sequent $\Gamma \to A$ containing k slashes is derivable in NLG_0, then it is also derivable in NLG_0^*.*

Proof: We prove this theorem with strong induction on the number of slashes $(/,\backslash)$ in a sequent. The base case is $k = 0$. The sequent must be an axiom. No R1$'$ or R1$''$ rules are used so Theorem 2.1 holds for $k = 0$. Now we assume the induction hypothesis:

IH(k): For all $i < k$, if some sequent $\Delta \to B$ containing i slashes is derivable in NLG_0, then $\Delta \to B$ is also derivable in NLG_0^*.

We have to prove that if some sequent $\Theta \to C$ containing k slashes is derivable in NLG_0, then it is also derivable in NLG_0^*.

Assume $\Theta \to C$ is derivable in NLG_0. Then there is a proof Π of $\Theta \to C$. We will show that $\Theta \to C$ is also derivable in NLG_0^*. We consider various possibilities for the last step in the proof of Π. We assume that proofs are constructed from top to bottom, i.e., from the axioms we try to reach the conclusion. The last step in a proof is the step that proves the final conclusion.

Case 1: Π is an axiom. Then it is derivable in NLG_0^* too.

Case 2: The last step in Π is an R2 rule or an R1 rule with $X \in Tp$. The proof is easy. The premises of the last step contain fewer slashes than $\Theta \to C$. The induction hypothesis tells that the premises are derivable in NLG_0^*. Use the NLG_0^* proofs of the premises and the last rule of Π to construct a proof of $\Theta \to C$ in NLG_0^*.

Case 3: The last step in Π is some R1 rule (e.g. R1′, the other case is symmetric) with $X \in BSTp$, but $X \notin Tp$.

$$\dfrac{\begin{array}{c}\vdots\\ (X, b) \to a\end{array}}{X \to a/b}\,[\mathrm{R1}']$$

The induction hypothesis tells that $(X, b) \to a$ is derivable in NLG_0^*. Consider the proof in NLG_0^* of $(X, b) \to a$. The last step in this proof is not an R1 rule because $(X, b) \notin Tp$. If $(X, b) \to a$ has been derived with an R2 rule, then we have various possibilities.

$$\dfrac{\dfrac{\cdots}{(X, b) \to a}\,[\mathrm{R2}]}{X \to a/b}\,[\mathrm{R1}']$$

The type b is not involved in the rule R2 (case 3a) or it is involved (case 3b).

Case 3a: If b is not involved, then R2 can be R2′ or R2″. The proof is similar for both cases. In the first case, the proof is of the form:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ X' \to y\end{array} \quad \begin{array}{c}\vdots\\ (Z[c], b) \to a\end{array}}{(Z[(c/y, X')], b) \to a}\,[\mathrm{R2}']}{Z[(c/y, X')] \to a/b}\,[\mathrm{R1}']$$

We can transform this proof into the following proof:

$$\dfrac{\begin{array}{c}\vdots\\ X' \to y\end{array} \quad \dfrac{\begin{array}{c}\vdots\\ (Z[c], b) \to a\end{array}}{Z[c] \to a/b}\,[\mathrm{R1}']}{Z[(c/y, X')] \to a/b}\,[\mathrm{R2}']$$

The sequents $Z[c] \to a/b$ and $X' \to y$ have fewer slashes than $Z[(c/y, X')] \to a/b$. The induction hypothesis tells that the two smaller sequents are provable in NLG_0^*. Therefore, $Z[(c/y, X')] \to a/b$ is also provable in NLG_0^*.

Case 3b: If b is involved in the R2 rule it must be the functor. X cannot be the functor because $X \notin Tp$. Type b is of the form $d\backslash c$.

$$\dfrac{\dfrac{\overset{\vdots}{X \to d} \quad \overset{\vdots}{c \to a}}{(X, d\backslash c) \to a}\,[\mathrm{R2}']}{X \to a/(d\backslash c)}\,[\mathrm{R1}']$$

The induction hypothesis says that $X \to d$ is derivable in NLG_0^*. Consider the proof of $X \to d$ in NLG_0^*. The last step in this proof must be an R2 step because $X \notin Tp$. The proof is of the following form:

$$\dfrac{\dfrac{\dfrac{\overset{\vdots}{X' \to y} \quad \overset{\vdots}{Z[e] \to d}}{Z[(e/y, X')] \to d}\,[\mathrm{R2}'] \qquad \overset{\vdots}{c \to a}}{(Z[(e/y, X')], d\backslash c) \to a}\,[\mathrm{R2}'']}{Z[(e/y, X')] \to a/(d\backslash c)}\,[\mathrm{R1}']$$

We can transform this into:

$$\dfrac{\overset{\vdots}{X' \to y} \qquad \dfrac{\dfrac{\overset{\vdots}{Z[e] \to d} \quad \overset{\vdots}{c \to a}}{(Z[e], d\backslash c) \to a}\,[\mathrm{R2}'']}{Z[e] \to a/(d\backslash c)}\,[\mathrm{R1}']}{Z[(e/y, X')] \to a/(d\backslash c)}\,[\mathrm{R2}']$$

The induction hypothesis says that $X' \to y$ and $Z[e] \to a/(d\backslash c)$ have proofs in NLG_0^*. Therefore $Z[(e/y, X')] \to a/(d\backslash c)$ has a proof in NLG_0^*. This completes the proof of Theorem 2.1. □

Theorem 2.2 *For all k, if some sequent $\Gamma \to A$ containing k slashes is derivable in NLG_0^*, then it is also derivable in NLG_0^{**}.*

Proof: Proof with strong induction on the number of slashes k. The base case $k = 0$ is trivial. Now we assume the induction hypothesis:

IH(k): For all $i < k$, if some sequent $\Delta \to B$ containing i slashes is derivable in NLG_0^*, then $\Delta \to B$ is also derivable in NLG_0^{**}.

We have to prove that if some sequent $\Theta \to C$ containing k slashes is derivable in NLG_0^*, then it is also derivable in NLG_0^{**}.

Assume $\Theta \to C$ is derivable in NLG_0^*. Then there is a proof Π of $\Theta \to C$ in NLG_0^*. We will show that $\Theta \to C$ is also derivable in NLG_0^{**}. We consider various possibilities for the last step in the proof of Π.

Case 1: Π is an axiom. Then it is derivable in NLG_0^{**} too.

Case 2: The last step in Π is an R1 rule or an R2 rule with $X \in Tp$. The premises of the last step contain fewer slashes than $\Theta \to C$. The induction hypothesis tells that the premises are derivable in NLG_0^{**}. Use the NLG_0^{**} proofs of the premises and the last rule of Π to construct a proof of $\Theta \to C$ in NLG_0^{**}.

Case 3: The last step in the proof is an R2 rule with $X \in BSTp$, but $X \notin Tp$.

$$\dfrac{\begin{matrix}\vdots & \vdots\\ X \to y & Y[x] \to z\end{matrix}}{Y[(x/y, X)] \to z}\;[\mathrm{R2}']$$

The induction hypothesis says that $X \to y$ has a proof in NLG_0^{**}. The last rule applied in the proof of this sequent cannot be an R1 rule, because $X \notin Tp$. So the last rule in the proof of $X \to y$ is an R2 rule, say R2′ (the R2″ case is similar). We have the following proof:

$$\dfrac{\dfrac{\begin{matrix}\vdots & \vdots\\ W \to v & Z[d] \to y\end{matrix}}{Z[(d/v, W)] \to y}\;[\mathrm{R2}'] \qquad \begin{matrix}\vdots\\ Y[x] \to z\end{matrix}}{Y[(x/y, Z[(d/v, W)])] \to z}\;[\mathrm{R2}']$$

where W is in Tp. We can transform it as follows:

$$\dfrac{\begin{matrix}\vdots\\ W \to v\end{matrix} \qquad \dfrac{\begin{matrix}\vdots & \vdots\\ Z[d] \to y & Y[x] \to z\end{matrix}}{Y[(x/y, Z[d])] \to z}\;[\mathrm{R2}']}{Y[(x/y, Z[(d/v, W)])] \to z}\;[\mathrm{R2}']$$

The sequents $W \to v$ and $Y[(x/y, Z[d])] \to z$ have proofs in NLG_0^{**} (induction hypothesis) and therefore $Y[(x/y, Z[(d/v, W)])] \to z$ has a proof in NLG_0^{**} too. End of proof of Theorem 2.2. □

Theorem 2.3 *Anything derivable in NLG_0^{**} is also derivable in NLG_0.*

Proof: This is trivial, any proof in NLG_0^{**} is also a proof in NLG_0. □

From Theorems 2.1, 2.2 and 2.3 we know that $\mathrm{NLG}_0 = \mathrm{NLG}_0^* = \mathrm{NLG}_0^{**}$. We introduce another calculus now, called A1–R2–R3–R4. This calculus is defined as follows:

$$[\mathrm{A1}]\quad x \to x \text{ , for } x \in Tp$$

$$\frac{w \to y \quad z \to x}{w \to x/(y\backslash z)}\,[\mathrm{R3'}] \qquad \frac{w \to y \quad z \to x}{w \to (z/y)\backslash x}\,[\mathrm{R3''}]$$

$$\frac{w \to y \quad z \to x}{w/x \to y/z}\,[\mathrm{R4'}] \qquad \frac{w \to y \quad z \to x}{x\backslash w \to z\backslash y}\,[\mathrm{R4''}]$$

$$\frac{w \to y \quad Y[x] \to z}{Y[(x/y, w)] \to z}\,[\mathrm{R2'}] \qquad \frac{w \to y \quad Y[x] \to z}{Y[(w, y\backslash x)] \to z}\,[\mathrm{R2''}]$$

for $Y \in BSTp$ and $w, x, y, z \in Tp$.

Figure 3: A1–R2–R3–R4

Theorem 2.4 *A1–R2–R3–R4* $= NLG_0^{**}$

Proof: Let us repeat the calculus NLG_0^{**} here.

$$[\mathrm{A1}]\quad x \to x \text{ , for } x \in Tp$$

$$\frac{(w, y) \to x}{w \to x/y}\,[\mathrm{R1'}] \qquad \frac{(y, w) \to x}{w \to y\backslash x}\,[\mathrm{R1''}]$$

$$\frac{w \to y \quad Y[x] \to z}{Y[(x/y, w)] \to z}\,[\mathrm{R2'}] \qquad \frac{w \to y \quad Y[x] \to z}{Y[(w, y\backslash x)] \to z}\,[\mathrm{R2''}]$$

for $Y \in BSTp$ and $w, x, y, z \in Tp$.

Figure 4: NLG_0^{**}

The last step in the proof of the premise of an R1 rule must be an R2 rule. Therefore we can replace R1′ and R1″ by the following rules:

$$\frac{w \to y \quad z \to x}{w \to x/(y\backslash z)}\,[\mathrm{R3'} = \mathrm{R2''} + \mathrm{R1'}] \qquad \frac{w \to y \quad z \to x}{w \to (z/y)\backslash x}\,[\mathrm{R3''} = \mathrm{R2'} + \mathrm{R1''}]$$

$$\frac{w \to y \quad z \to x}{w/x \to y/z}\,[\mathrm{R4'} = \mathrm{R2'} + \mathrm{R1'}] \qquad \frac{w \to y \quad z \to x}{x\backslash w \to z\backslash y}\,[\mathrm{R4''} = \mathrm{R2''} + \mathrm{R1''}]$$

This proves Theorem 2.4. □

Observe that the antecedent of the conclusion of an R2 rule is not in Tp. On the other hand, the antecedents of the premises and the conclusions of R3 and R4 rules are in Tp. Hence, a proof in A1–R2–R3–R4 of a sequent with an antecedent in Tp contains R3 and R4 rules only.

3 Recognition for non-associative categorial grammar

We have come to a point now where we can remove the brackets from our calculi. We define the bracket-free calculus NLG_1. The rules of NLG_1 are:

$$[\mathrm{A1}]\ \ x \rightarrow x \text{ , for } x \in Tp$$

$$\frac{w \rightarrow y \quad z \rightarrow x}{w \rightarrow x/(y\backslash z)}\,[\mathrm{R3'}] \qquad \frac{w \rightarrow y \quad z \rightarrow x}{w \rightarrow (z/y)\backslash x}\,[\mathrm{R3''}]$$

$$\frac{w \rightarrow y \quad z \rightarrow x}{w/x \rightarrow y/z}\,[\mathrm{R4'}] \qquad \frac{w \rightarrow y \quad z \rightarrow x}{x\backslash w \rightarrow z\backslash y}\,[\mathrm{R4''}]$$

$$\frac{x \rightarrow y \quad \Delta, w, \Delta' \rightarrow z}{\Delta, w/y, x, \Delta' \rightarrow z}\,[\mathrm{R5'}] \qquad \frac{x \rightarrow y \quad \Delta, w, \Delta' \rightarrow z}{\Delta, x, y\backslash w, \Delta' \rightarrow z}\,[\mathrm{R5''}]$$

for $\Delta, \Delta' \in STp$ and $w, x, y, z \in Tp$.

Figure 5: NLG_1

This calculus differs from A1–R2–R3–R4 because the brackets in the rules R2$'$ and R2$''$ are erased. This results in two new rules R5$'$ and R5$''$.

There is a one-to-one correspondence between proofs in A1–R2–R3–R4 and NLG_1. We can prove the following lemma's:

Lemma 3.1 *If a bracketed sequent $X \rightarrow z$ is derivable in A1–R2–R3–R4, then $yield(X) \rightarrow z$ is derivable in NLG_1.*

Proof: A proof in A1–R2–R3–R4 has the following shape:

$$\dfrac{\overset{\vdots\ R3\text{-}R4}{x_1 \rightarrow w_1} \qquad \dfrac{\overset{\vdots\ R3\text{-}R4}{x_2 \rightarrow w_2} \qquad \dfrac{\overset{\vdots\ R3\text{-}R4}{x_3 \rightarrow w_3} \qquad \overset{\vdots}{Y_3 \rightarrow z}}{Y_2 \rightarrow z}\,[\mathrm{R2}]}{Y_1 \rightarrow z}\,[\mathrm{R2}]}{Y_0 \rightarrow z}\,[\mathrm{R2}]$$

The $Y_i \rightarrow z$ premises are the only premises containing brackets. When we leave out those brackets, we immediately obtain an NLG_1 proof. $\square$

Lemma 3.2 *If a sequent $Y \to z$ is derivable in NLG_1, then there is an X such that $yield(X) = Y$ and $X \to z$ is derivable in A1–R2–R3–R4.*

Proof: If we have an NLG_1 proof it has the following shape:

$$\dfrac{\begin{matrix}\vdots\ R3\text{-}R4\\ x_1 \to w_1\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\ R3\text{-}R4\\ x_2 \to w_2\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\ R3\text{-}R4\\ x_3 \to w_3\end{matrix} \qquad \begin{matrix}\vdots\\ Y_3 \to z\end{matrix}}{Y_2 \to z}\,[\mathrm{R5}]}{Y_1 \to z}\,[\mathrm{R5}]}{Y_0 \to z}\,[\mathrm{R5}]$$

It is easy to add brackets to the Y_i such that we obtain an A1–R2–R3–R4 proof. □

Lemma's 3.1 and 3.2 enable us to remove the brackets in the definition of grammaticality as well.

$L(G)$ has been defined as the set of strings $w \in \Sigma^*$, $w = a_1 \dots a_n$ such that for some *bracketed* string of types X, $yield(X) = A_1, \dots, A_n$, $\langle a_i, A_i \rangle \in Lex$ $(1 \leq i \leq n)$ and $X \to s$ is derivable in NLG_0.

But the language can be defined bracket-free now: $L'(G)$ is the set of strings $w \in \Sigma^*$, $w = a_1 \dots a_n$ such that for some string of types $A_1, \dots, A_n$, $\langle a_i, A_i \rangle \in Lex$ $(1 \leq i \leq n)$ and $A_1, \dots, A_n \to s$ in NLG_1.

Lemma 3.3 *For every lexicon G, $L'(G) = L(G)$.*

Proof: Follows from Lemma's 3.1 and 3.2, Theorem 2.4 and the fact that $\mathrm{NLG}_0 = \mathrm{NLG}_0^{**}$. □

Lemma 3.4 *For all $x, y \in Tp$, we can check in time $\mathcal{O}((|x| + |y|)^2)$ whether $x \to y$ is provable in NLG_1.*

Proof: The antecedents of the premises and of the conclusions of the R3 and R4 rules have length one. The antecedent of the conclusion of an R5 rule has length bigger than one. Therefore, the proof of $x \to y$ consists of R3 and R4 rules only. The subformula property holds for the "R3–R4 calculus". Let n be the number of slashes in x plus the number of slashes in y. n is linear in $|x| + |y|$. We have to compute at most $\mathcal{O}(n^2)$ times whether some $a \to b$ is derivable (because a and b are subformulas of x and y). We *memoize* (Cormen *et al.*, 1990, pp. 312–314) the results of the attempts to prove something: the first time we have to compute $a \to b$ the result of the computation is stored in a table. If we have to compute it again later we look up the answer in the table. The search space is a graph with nodes labeled $x \to y$. As said, there are $\mathcal{O}(n^2)$ nodes. Every node has at most four outgoing arcs: at most two rules of R3–R4 are applicable, so we have to prove at most 4 premises. Therefore, the number of arcs is $\mathcal{O}(4n^2) = \mathcal{O}(n^2)$. The algorithm is a depth first traversal of the graph. Because of memoization, every arc is traversed only once. The algorithm takes time $\mathcal{O}(n^2)$ at most. □

Theorem 3.5 *The recognition problem for non-associative Lambek categorial grammar can be reduced to context-free grammar recognition in polynomial time.*

Proof: We take the new definition of categorial languages: $L'(G)$ is the set of strings $w \in \Sigma^*$, $w = a_1 \dots a_n$ such that for some string of types $A_1, \dots, A_n$, $\langle a_i, A_i \rangle \in Lex$ $(1 \leq i \leq n)$ and $A_1, \dots, A_n \to s$ in NLG_1. The symbol s is the distinguished type of the grammar ($s \in Pr$). Proofs in NLG_1 look like:

$$\dfrac{\begin{array}{c}\vdots\ R3\text{-}R4\\ x_1 \to w_1\end{array} \qquad \dfrac{\begin{array}{c}\vdots\ R3\text{-}R4\\ x_2 \to w_2\end{array} \qquad \dfrac{\begin{array}{c}\vdots\ R3\text{-}R4\\ x_3 \to w_3\end{array} \qquad \begin{array}{c}\dfrac{x_n \to w_n \quad s \to s}{}\ [R5]\\ \vdots\\ Y_3 \to s\end{array}}{Y_2 \to s}\ [R5]}{Y_1 \to s}\ [R5]}{Y_0 \to s}\ [R5]$$

The sequence $s, \dots, Y_3, Y_2, Y_1, Y_0$ can be seen as a derivation in a context-free grammar. Given a lexicon, we construct a context-free grammar that generates the same language as the categorial grammar. The context-free grammar consists of binary and unary grammar rules. The start symbol is s. The binary rules simulate rewriting the symbols $s, \dots, Y_3, Y_2, Y_1, Y_0$. The unary rules simulate the lexical type assignment.

The binary rules are of the form $w \Rightarrow w/y, x$ and $w \Rightarrow x, y\backslash w$. Let the variables x, w/y, and $y\backslash w$ range over all possible subtypes in the lexicon. The rule $w \Rightarrow w/y, x$ (or $w \Rightarrow x, y\backslash w$) is added to the grammar when the sequent $x \to y$ is derivable in NLG_1. The unary rules (for lexical type assignment) are of the form $A \Rightarrow a$ with $\langle a, A \rangle \in Lex$.

The number of subtypes in the lexicon is linear in the size of the lexicon. The number of binary rules is kwadratic in the number of subtypes. Therefore, the size of the grammar is kwadratic in the size of the lexicon. Construction of the grammar takes polynomial time because we can compute $x \to y$ in polynomial time. $\Box$

The time complexity of context-free grammar recognition is $\mathcal{O}(|G|n^3)$ where $|G|$ is the size of the grammar and n the length of the input sentence (Sippu and Soisalon-Soininen, 1988, p. 147). Via construction of the context-free grammar, we have a polynomial time algorithm for recognition in the non-associative Lambek categorial grammar. After construction of the grammar, the time complexity of recognition is cubic in the length of the string and kwadratic in the size of the lexicon.

A polynomial time algorithm for deciding provability in NLG_1 can be given too. Because we do not have a lexicon anymore, the number of possible types is infinite. We cannot use a context-free grammar in the style described here because it would have infinitely many rules. But instead of computing the grammar in advance we can compute grammar rules "on the fly". We try to combine adjacent types according to the schemes $w \Rightarrow w/y, x$ and $w \Rightarrow x, y\backslash w$ and compute whether $x \to y$ is derivable in R3–R4.

Acknowledgements

We want to thank Kees Vermeulen for proofreading an earlier version of this paper.

References

Erik Aarts. Parsing second order Lambek grammar in polynomial time. In Paul Dekker and Martin Stokhof, editors, *Proceedings of the Ninth Amsterdam Colloquium*, pages 35–45. Institute for Logic, Language and Computation, University of Amsterdam, December 1993.

Erik Aarts. Proving theorems of the second order Lambek calculus in polynomial time. *Studia Logica*, to appear.

Wojciech Buszkowski. Generative capacity of the nonassociative Lambek calculus. *Bull. Pol. Acad. Scie. Math.*, 34:507–516, 1986.

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms.* MIT Press, 1990.

Theo M.V. Janssen. On Properties of the Zielonka-Lambek Calculus. In Paul Dekker and Martin Stokhof, editors, *Proceedings of the Eighth Amsterdam Colloquium*, pages 303–308. Institute for Logic, Language and Computation, University of Amsterdam, December 1991.

Maciej Kandulski. The non-associative Lambek calculus. In W. Buszkowski, W. Marciszewski, and J. van Benthem, editors, *Categorial Grammar*, Linguistic and Literary Studies in Eastern Europe (LLSEE), pages 141–151. John Benjamins Publishing Company, Amsterdam–Philadelphia, 1988.

Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, pages 154–169, 1958.

Seppo Sippu and Eljas Soisalon-Soininen. *Parsing Theory, Vol. 1: Languages and Parsing.* EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1988.

Kees Trautwein. Een parseeralgoritme voor de niet associatieve Lambek calculus [a parsing algorithm for the non-associative lambek calculus]. Master's thesis, University of Amsterdam, August 1991.