

A solution to the emptiness problem for Lambek calculus and some of its extensions

Matthijs Melissen (info@matthijsmelissen.nl), CKI, Universiteit Utrecht

May 12, 2008

Abstract

This paper presents a way of solving the emptiness problem, that is the problem of telling whether a given language contains no strings at all, for associative and nonassociative Lambek calculus, and the logic of pure residuation in general. The correctness of this method is proved by first converting the Lambek grammar to context-free grammar, and then solving the emptiness problem for context-free grammar.

1 Introduction

One can ask oneself whether a given set of words has a subset with which one can make a grammatical sentence. Consider for example the sets of words {dog, crosses, street, the} and {cat, triangle, street, the}. With the first set, it is possible to generate such a sentence ('the dog crosses the street'), while one cannot do this with the second set. This paper presents a general way of solving these kinds of problems within categorial grammars like Lambek calculus and the logic of pure residuation. First I will formally introduce Lambek calculus and the emptiness problem. Secondly, I show a method of solving the emptiness problem for Lambek calculus, and prove the correctness of this method. Then a generalization of this solution to the logic of pure residuation is presented. Finally I draw some conclusions and give some ideas for further research.

2 Preliminaries

2.1 Non-associative Lambek calculus

Now we will introduce the *non-associative Lambek calculus NL* ([5]), which we use to reason about natural language and grammaticality. Lambek calculus is a deductive system. A sentence is grammatical if and only if the sentence can be derived in the deductive system. 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 a and b are types, $a \bullet b$, $a \backslash b$

(pronounced 'a under b') and a/b (pronounced 'a over b') are also types. Types will be denoted with italic letters a, b, \dots . Intuitively we can see one or more words with type $a \backslash b$ 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 asks for a phrase of type b on its right-hand side, before returning a b type phrase. Finally the \bullet operator can be seen as concatenation of words.

We use one axiom, called identity:

$$a \rightarrow a$$

Furthermore we use the following rules (with a double line we mean derivability in both directions), called respectively transitivity and residuation.

$$\frac{a \rightarrow b \quad b \rightarrow c}{a \rightarrow c} \text{ (TRANS)} \quad \frac{b \rightarrow a \backslash c}{a \bullet b \rightarrow c} \text{ (RES)}$$

Example 1. $\text{NL} \vdash a \rightarrow b/(a \backslash b)$. Proof:

$$\frac{a \backslash b \rightarrow a \backslash b}{a \bullet (a \backslash b) \rightarrow b} \text{ (RES)}$$
$$\frac{a \bullet (a \backslash b) \rightarrow b}{a \rightarrow b/(a \backslash b)} \text{ (RES)}$$

2.2 Natural deduction for NL

We can also present NL in a natural deduction format.

First we will define what is a *structure*. A structure might be empty, consist of a single type, or have

the form $\Gamma \circ \Delta$ where Γ and Δ are structures. We will denote structures with capital greek letters. We define the *yield* of a formula as $\text{yield}(a) = a$ if a is a single type, $\text{yield}(a \circ b) = \text{yield}(a)\text{yield}(b)$ and $\text{yield}(a \Rightarrow b) = \text{yield}(a) \Rightarrow \text{yield}(b)$. A *sequent* has the form $\Gamma \rightarrow A$, where Γ is a nonempty sequence of types. The notation $\frac{\Gamma[\Theta]}{\Gamma[\Delta]}$ means that in the structure Γ , the substructure Δ can be replaced by Θ .

Again we use only one axiom, namely the identity $a \rightarrow a$. Furthermore we use the following inference rules:

$$\begin{array}{c} \frac{\Gamma \rightarrow a/b \quad \Delta \rightarrow b}{\Gamma \circ \Delta \rightarrow a} /E \\ \frac{\Gamma \rightarrow b \quad \Delta \rightarrow b \backslash a}{\Gamma \circ \Delta \rightarrow a} \backslash E \\ \frac{\Delta \rightarrow a \bullet b \quad \Gamma[a \circ b] \rightarrow c}{\Gamma[\Delta] \rightarrow c} \bullet E \end{array} \quad \begin{array}{c} \frac{\Gamma \circ b \rightarrow a}{\Gamma \rightarrow a/b} /I \\ \frac{b \circ \Gamma \rightarrow a}{\Gamma \rightarrow b \backslash a} \backslash I \\ \frac{\Gamma \rightarrow a \quad \Delta \rightarrow b}{\Gamma \circ \Delta \rightarrow a \bullet b} \bullet I \end{array}$$

In all rules, we require the left-hand side of the arrow to be nonempty.

We write $\mathbf{NL} \vdash \Gamma \rightarrow a$ if the sequent $\Gamma \rightarrow a$ is derivable in the nonassociative Lambek calculus.

2.3 Lambek Grammar

Now we have presented the Lambek calculus, we will define a *Lambek grammar*. Intuitively, a grammar can be seen as a method to decide which sentences are, and which sentences are not in the language. A Lambek grammar consists of a set of symbols - which correspond to words in natural language - a type of the goal formula (i.e. the type corresponding to a entire sentence) and a mapping which assigns one or more types to every word, called the *dictionary*. Now we present a Lambek grammar formally.

Definition 2 (Lambek grammar). We assume that a finite alphabet Σ and a distinguished type D are given. A *Lambek grammar* is parametrized by Σ , D and a mapping f such that for all $t \in \Sigma$, $f(t) \subset T$ and $f(t)$ is finite, where T is the set of all types of the Lambek calculus. In this paper, we restrict ourselves to the case where D is an atomic formula. The *language generated by the Lambek grammar* $\mathcal{L}(\Sigma, D, f)$ is defined as the set of all expressions $t_1 \dots t_n$ over the alphabet Σ for which there exists a derivable sequent with yield $B_1 \dots B_n \rightarrow D$ such that $B_i \in f(t_i)$ for all $i \leq n$. We also write $t_1 \dots t_n \in \mathcal{L}$ if $t_1 \dots t_n$ is generated by \mathcal{L} .

Now we will show an example natural deduction derivation of a natural language sentence. To prove grammaticality of a sentence, we first turn the sentence into a structure by adding brackets and putting \circ symbols in between the words. Then we show that this structure is of type s .

Example 3. Let us derive the sentence 'Alice sees the house', assuming the following types: $f(\text{Alice}) = \{np\}$, $f(\text{sees}) = \{(np \backslash s)/np\}$, $f(\text{the}) = \{np/n\}$, and $f(\text{house}) = \{n\}$. For the whole sentence we then obtain the type $np \circ (((np \backslash s)/np) \circ ((np/n) \circ n))$, so the sentence can be derived as follows:

$$\begin{array}{c} \frac{np/n \rightarrow np/n \quad n \rightarrow n}{(np \backslash s)/np \rightarrow (np \backslash s)/np \quad (np/n) \circ n \rightarrow np} \\ \frac{np \rightarrow np \quad ((np \backslash s)/np) \circ ((np/n) \circ n) \rightarrow np \backslash s}{np \circ (((np \backslash s)/np) \circ ((np/n) \circ n)) \rightarrow s} \end{array}$$

2.4 Emptiness problem

Definition 4 (Emptiness problem). Recall the question whether it is possible to make a sentence from a given set of words, presented in the introduction. This can be seen as an instance of the *emptiness problem*. More general, we can say that the emptiness problem is the problem of deciding whether a given language is empty or $L(\mathcal{G}) = \emptyset$, i.e. whether the corresponding grammar does not generate any *string* (a list of zero or more symbols).

3 The emptiness problem for NL

3.1 A solution

In this section I will present a method to determine whether a nonassociative Lambek grammar generates the empty language. First I will sketch intuitively what is hard about this. Next I will give a construction to determine whether a language is empty, and finally I prove that this method is correct.

Definition 5. We can say that the *emptiness problem for Lambek calculus* it is the problem of determining whether $\mathcal{L}(\Sigma, D, f) = \emptyset$, given an alphabet Σ , a designated type D and a mapping f .

Example 6. i Consider the language $\mathcal{L}(\{a, b\}, s, f)$, where $f(a) = \{p \backslash (s/p)\}$ and $f(b) = \{p\}$. It is easy to see that this language is nonempty: it

holds that $(p \circ (p \setminus (s/p))) \circ p \rightarrow s$, so **bab** is a valid string in the language.

- ii Now consider the language $\mathcal{L}(\{\mathbf{a}\}, s, f)$, where $f(\mathbf{a}) = \{s/s\}$. It is clear that this language is empty.

However, in general it is not so easy to check by hand whether a language is empty. Note that a symbol might occur more than once in a string. It is not so easy to come up with an upper bound for the shortest string generated by the language. It might well be possible that a grammar has only a few types in the dictionary, but still only generates very long strings. Note that in particular it is not possible to check every possible arrangement of types. Of course this procedure will find a derivation if there is one, but because strings can be arbitrarily long, the process will never terminate in case there exists no derivation.

Definition 7. We define the *length* of a type as the total number of primitive type occurrences in the type:

$$\|p_i\| := 1 \quad \|a \bullet b\| = \|a/b\| = \|a \setminus b\| := \|a\| + \|b\|$$

Definition 8. Assume a Lambek grammar $\mathcal{L}(\Sigma, D, f)$ is given. Notice that the number of atoms used in the definition of \mathcal{L} is finite. Below we will consider only types consisting of these atoms. Let m be the length of the largest type in the dictionary f , that is the smallest number such that $m \geq \|t\|$ for all $t \in \bigcup_{\sigma \in \Sigma} f(\sigma)$. We define the sequence $S_{\mathcal{L}}$ in the following way:

$$\begin{aligned} S_{\mathcal{L}}(0) &= \{y \mid \langle x, y \rangle \in f\} \\ S_{\mathcal{L}}(n+1) &= S_{\mathcal{L}}(n) \\ &\cup \{a \mid \mathbf{NL} \vdash b \circ c \rightarrow a; \\ &\quad b, c \in S_{\mathcal{L}}(n); \|a\| \leq m\} \\ &\cup \{a \mid \mathbf{NL} \vdash b \rightarrow a; \\ &\quad b \in S_{\mathcal{L}}(n); \|a\| \leq m\} \end{aligned}$$

In other words, we start with all types in our dictionary, and iteratively add all types which can be derived from one or two old types of length no longer than the length of the largest type in the dictionary.

Definition 9. The *limit* of f , if it exists, is the smallest n such that $f(n) = f(n+1)$.

Theorem 10 (The emptiness problem for **NL**). *The language generated by a nonassociative Lambek grammar $\mathcal{L}(\Sigma, D, f)$ is nonempty if and only if D is an element of the limit of $S_{\mathcal{L}}$.*

This theorem will be proved in section 3.4.

Example 11. Let us look at the following nonassociative Lambek grammar: $\mathcal{L}(\{\mathbf{a}, \mathbf{b}\}, s, f)$, where $f(\mathbf{a}) = np \setminus (s/np)$ and $f(\mathbf{b}) = np$.

Now $\|np \setminus (s/np)\| = 3$ and $\|np\| = 1$, so $m = 3$. Then it follows that $np, np \setminus (s/np) \in S_{\mathcal{L}}(0)$, so $s/np \in S_{\mathcal{L}}(1)$, and therefore $s \in S_{\mathcal{L}}(2)$. Then s is an element of the limit of $S_{\mathcal{L}}$ as well, so this language is nonempty. This is correct, because we can easily see that **bab** is a valid string in the language.

Example 12. Now we look at this grammar: $\mathcal{L}(\{\mathbf{a}, \mathbf{b}\}, p, f)$, where $f(\mathbf{a}) = p/q$ and $f(\mathbf{b}) = q/p$.

In this case $\|p/q\| = 2$ and $\|q/p\| = 2$, so $m = 2$. We see that $S_{\mathcal{L}}(0) = \{p/q, q/p\}$, and $S_{\mathcal{L}}(1) = S_{\mathcal{L}}(0)$, so $S_{\mathcal{L}}(n) = S_{\mathcal{L}}(n-1)$ for $n \geq 1$. Then it follows that p is not an element of the limit $S_{\mathcal{L}}$. Therefore the language is empty.

3.2 Lambek calculus and context-free grammars

Now we have seen some examples of the correctness of theorem 10, but we have not yet seen a general proof. This will be presented now. The proof consists of three parts. First I translate the Lambek grammar to a context-free grammar. Then I show how to solve the emptiness problem for context-free grammar, which is much easier than solving the problem for Lambek grammar. Finally I show how this leads to the construction presented above.

I will start by defining what is a context-free grammar [1].

Definition 13. A *context-free grammar* \mathcal{G} has the form

$$\mathcal{G}(\Sigma, \mathcal{W}, S, R),$$

where Σ is a finite set of elements called terminals, \mathcal{W} is a finite set of elements called non-terminal characters, disjoint with Σ , S is the start symbol, and R is a finite relation of rules from \mathcal{W} to $(\mathcal{W} \cup \Sigma)^*$.

For all strings u, v we write $u \Rightarrow v$ iff $\exists(\alpha, \beta) \in R, u_1, u_2 \in (\mathcal{W} \cup \Sigma)^*$ such that $u = u_1 \alpha u_2$ and $v = u_1 \beta u_2$. We write $u \Rightarrow^* v$ if there exists $u_1, u_2, \dots, u_k \in (\mathcal{W} \cup \Sigma)^*$ such that $u \Rightarrow u_1 \Rightarrow u_2 \dots \Rightarrow u_k \Rightarrow v$. Now we define the language generated by the grammar $\mathcal{G}(\Sigma, \mathcal{W}, R, S)$ as $L(\mathcal{G}) = \{w \in \Sigma^* : S \Rightarrow^* w\}$.

Now we show how to translate a given nonassociative Lambek grammar into a context-free grammar.

Although equivalence between nonassociative Lambek grammar and context-free grammar has been established earlier ([3]), we will make use of the construction presented in [2].

Lemma 14. *Assume a nonassociative Lambek grammar $\mathcal{L}(\Sigma, D, f)$ is given. We can construct a context-free grammar $\mathcal{G}(\Sigma', \mathcal{W}, S, R)$ generating the same language as follows. Again we define m as the smallest number such that $m \geq \|t\|$ for all $t \in \bigcup_{\sigma \in \Sigma} f(\sigma)$. First we set $\Sigma' = \Sigma$. Then we define the set of auxiliary symbols \mathcal{W} :*

$$\mathcal{W} := \{a \in T \mid \|a\| \leq m\}$$

Furthermore we set $S = D$. Finally the set \mathcal{R} of rules is defined in the following way:

$$\begin{aligned} \mathcal{R} &:= \{b \Rightarrow t \mid t \in \Sigma \text{ and } b \in f(t)\} \\ &\cup \{a \Rightarrow bc \mid a, b, c \in \mathcal{W} \text{ and} \\ &\quad \mathbf{NL} \vdash b \circ c \rightarrow a\} \\ &\cup \{a \Rightarrow b \mid a, b \in \mathcal{W} \text{ and } \mathbf{NL} \vdash b \rightarrow a\} \end{aligned}$$

Example 15. Consider the following nonassociative Lambek grammar $\mathcal{L}(\{\mathbf{a}, \mathbf{b}\}, s, f)$, where $f(\mathbf{a}) = p$ and $f(\mathbf{b}) = p \setminus s$.

Note that $m = 2$. We translate \mathcal{L} to the context-free grammar $(\{a, b\}, \mathcal{W}, s, R)$, where \mathcal{W} is defined as follows:

$$\mathcal{W} = \{p, s, "p \setminus p", "p \setminus s", "s \setminus p", "s \setminus s", "p/p", "p/s", "p \bullet p", "s/p", "s/s", "p \bullet s", "s \bullet p", "s \bullet s\},$$

and R contains the following rules:

$$\begin{array}{ll} p \Rightarrow a & "p \setminus s" \Rightarrow b \\ s \Rightarrow "s/s" s & s \Rightarrow s "s \setminus s" \\ s \Rightarrow "s/p" p & s \Rightarrow p "p \setminus s" \\ p \Rightarrow "p/s" s & p \Rightarrow s "s \setminus p" \\ p \Rightarrow "p/p" p & p \Rightarrow p "p \setminus p" \end{array}$$

Note that compound symbols like " $p \setminus s$ " should still be seen as only one symbol.

3.3 The emptiness problem for context-free grammars

Now we will present a method to solve the emptiness problem of the generated context-free grammar. Solutions to this problem exist already ([1]), but we will present our own method, which we can easier translate back into Lambek grammar.

Definition 16. Assume a context-free grammar $\mathcal{G}(\Sigma, \mathcal{W}, S, \mathcal{R})$ as obtained by lemma 14, where all

rules have either one or two auxiliary symbols on the right-hand side, or one terminal symbol. We define the sequence $S_{\mathcal{G}}$ as follows:

$$\begin{aligned} S_{\mathcal{G}}(0) &= \{b \mid b \Rightarrow t \in \mathcal{R}\} \\ S_{\mathcal{G}}(n+1) &= S_{\mathcal{G}}(n) \\ &\cup \{A \mid a \Rightarrow bc \in \mathcal{R}; b, c \in S_{\mathcal{G}}(n)\} \\ &\cup \{A \mid a \Rightarrow b \in \mathcal{R}; b \in S_{\mathcal{G}}(n)\} \end{aligned}$$

That is, $S_{\mathcal{G}}$ will initially contain all nonterminals which can be rewritten in one step to a terminal, and we iteratively add to it all nonterminals which can be rewritten in one step to a nonterminal which is already in $S_{\mathcal{G}}$.

Lemma 17 (The emptiness problem for context-free grammar). *The language generated by $\mathcal{G}(\Sigma, \mathcal{W}, S, \mathcal{R})$ is nonempty if and only if S is an element of the limit of $S_{\mathcal{G}}$.*

Proof. Because Σ is finite, the limit of $S_{\mathcal{G}}$ clearly always exists.

[\Rightarrow] Assume the language generated by $\mathcal{G}(\Sigma, \mathcal{W}, d, \mathcal{R})$ is nonempty. By induction on the length of the derivation: if $d \Rightarrow t \in \mathcal{R}$ for some $t \in \Sigma$, d will be added to $S_{\mathcal{G}}$ by the base case of the definition. Now assume $d \Rightarrow b \in \mathcal{R}$ such that $\mathcal{G}(\Sigma, \mathcal{W}, b, \mathcal{R})$ is nonempty. By induction hypothesis, $b \in S_{\mathcal{G}}$. Then d will be added to $S_{\mathcal{G}}$ as well, by the third line of recursion step of the definition. Finally assume $d \Rightarrow bc \in \mathcal{R}$ such that $\mathcal{G}(\Sigma, \mathcal{W}, b, \mathcal{R})$ and $\mathcal{G}(\Sigma, \mathcal{W}, c, \mathcal{R})$ are nonempty. Again by induction hypothesis, b and c are in $S_{\mathcal{G}}$. Now d will be added to $S_{\mathcal{G}}$ by the second line of the recursion step of the definition. Therefore d is an element of the limit of $S_{\mathcal{G}}$.

[\Leftarrow] Consider the language generated by $\mathcal{G}(\Sigma, \mathcal{W}, d, \mathcal{R})$. Assume d is an element of the limit of $S_{\mathcal{G}}$. Induction on the number of the step in which d has been added. Assume $d \in S_{\mathcal{G}}(0)$. Then $d \Rightarrow t \in \mathcal{R}$ for $t \in \Sigma$, so the language is nonempty. Now assume d has been added to $S_{\mathcal{G}}$ in step $n+1$. This means $d \in S_{\mathcal{G}}(n+1)$ and not $d \in S_{\mathcal{G}}(n)$. Then either $d \in \{a \mid a \Rightarrow bc \in \mathcal{R}; b, c \in S_{\mathcal{G}}(n)\}$ or $d \in \{a \mid a \Rightarrow b \in \mathcal{R}; b \in S_{\mathcal{G}}(n)\}$. In the first case, by induction hypothesis $\mathcal{G}(\Sigma, \mathcal{W}, b, \mathcal{R})$ and $\mathcal{G}(\Sigma, \mathcal{W}, c, \mathcal{R})$ are nonempty. Because $d \Rightarrow bc$, $\mathcal{G}(\Sigma, \mathcal{W}, d, \mathcal{R})$ is nonempty as well. In the second case, $\mathcal{G}(\Sigma, \mathcal{W}, b, \mathcal{R})$ is nonempty by induction hypothesis, so $d \Rightarrow b$, so $\mathcal{G}(\Sigma, \mathcal{W}, d, \mathcal{R})$ is also nonempty. We can conclude that in every case, the language generated by $\mathcal{G}(\Sigma, \mathcal{W}, d, \mathcal{R})$ is nonempty. \square

Example 18. Consider the context-free grammar from example 15. Now it holds that $S_{\mathcal{G}}(0) = \{p, (p \setminus s)\}$. Then $S_{\mathcal{G}}(1) = \{p, (p \setminus s), s\}$. We see that s is an element of the limit of $S_{\mathcal{G}}$, so the language is nonempty.

3.4 Proof of the main theorem

Now we know how to translate **NL** into a context-free grammar, and how to solve the emptiness problem for context-free grammars, we can prove that the method to solve the emptiness problem for **NL** works correctly.

Proof of theorem 10. First let us check that $S_{\mathcal{L}}$ always has a limit. Because we consider only types of length smaller than m using a finite number of primitive types, the number of formulas that possibly can be added to $S_{\mathcal{L}}$ is finite as well. Therefore $S_{\mathcal{L}}$ has indeed always a limit.

Now we prove that if we translate the Lambek grammar \mathcal{L} to the context-free grammar \mathcal{G} by the procedure of lemma 14, it holds that $S_{\mathcal{L}} = S_{\mathcal{G}}$. We prove this by induction. First we see that $\{y \mid \langle x, y \rangle \in f\} = \{b \mid b \Rightarrow t \in \mathcal{R}\}$, so $S_{\mathcal{L}}(0) = S_{\mathcal{G}}(0)$. Now note that if $\|a\| \leq m$ then $a \in \mathcal{W}$ and that by induction hypothesis $c \in S_{\mathcal{G}}(n-1)$ if and only if $c \in S_{\mathcal{L}}(n-1)$. Therefore it holds that

$$\begin{aligned} & \{a \mid NL \vdash b \circ c \rightarrow a; b, c \in S_{\mathcal{L}}(n-1); \\ & \quad \|a\| \leq m\} \\ = & \{a \mid a \Rightarrow bc \in \mathcal{R}; b, c \in S_{\mathcal{G}}(n-1)\} \end{aligned}$$

and that

$$\begin{aligned} & \{a \mid NL \vdash b \rightarrow a, b \in S_{\mathcal{L}}(n-1), \|a\| \leq m\} \\ = & \{a \mid a \Rightarrow b \in \mathcal{R}, b \in S_{\mathcal{G}}(n-1)\}, \end{aligned}$$

so we can conclude that $S_{\mathcal{G}} = S_{\mathcal{L}}$. Because d in the Lambek grammar is translated to S in the context-free grammar, we can conclude by lemma 17 that \mathcal{L} is nonempty if and only if d is an element of the limit of $S_{\mathcal{L}}$. \square

4 Extensions to other calculi

4.1 The associative Lambek calculus **L**

Definition 19. The *associative Lambek calculus* **L** ([4]) consists of the same axioms and rules as the non-associative variant, except that we add associativity as a rule:

$$\frac{\Gamma[a \circ (b \circ c)] \rightarrow s}{\Gamma[(a \circ b) \circ c] \rightarrow s} \mathbf{A}$$

Theorem 20. *Theorem 10 works for the associative Lambek calculus **L** as well.*

Proof. The proof runs exactly the same as the corresponding proof for **NL**, except that we replace derivability in **NL** with derivability in **L**, and that we use [8] to prove the correctness of the conversion from Lambek calculus to context-free grammar. \square

4.2 Logic of pure residuation

We can generalize our theorem to logics of pure residuation (**LPR**) [2] as well. Instead of only binary operators, we consider now operators of an arbitrary number of arguments. To do this, first we define a set \mathcal{M} of so called modes. Furthermore we define a function δ which assigns a number of arguments ('arity') to every mode. Every mode $g \in \mathcal{M}$ defines $m+1$ connectives of m arguments each: a product operator g_{\bullet} , and m so called implications $\{g_{\rightarrow}^i \mid 1 \leq i \leq m\}$. Now we can define the system. Again we use identity as the only axiom, and transitivity as rule. Furthermore we add the residuation rule

$$\frac{g_{\bullet}(a_1, \dots, a_{\delta(g)}) \rightarrow b}{a_i \rightarrow g_{\rightarrow}^i(a_1, \dots, a_{i-1}, b, a_{i+1}, \dots, a_{\delta(g)})}$$

for every $g \in \mathcal{M}$ and $i \leq \delta(g)$.

Note that the residuation rule in section 2.1 is an instance of this definition, so **NL** is an instance of **LPR**. Many other calculi also can be seen as examples of **LPR**, among which **NL** \diamond ([6]). Note that **L** is not a logic of pure residuation, because the associativity rule does not hold in **LPR**.

Definition 21. Assume a **LPR**-language $\mathcal{L}(\Sigma, D, f)$ is given. Again we will only consider the types consisting of the (finite) set of primitive types used in the definition of \mathcal{L} . Let m be the smallest number such that $m \geq \|t\|$ for all $t \in \bigcup_{\sigma \in \Sigma} f(\sigma)$. We define the sequence $S_{\mathcal{L}}$ as follows:

$$\begin{aligned} S_{\mathcal{L}}(0) &= \{y \mid \langle x, y \rangle \in f\} \\ S_{\mathcal{L}}(n+1) &= S_{\mathcal{L}}(n) \\ &\cup \{a \mid \mathbf{LPR} \vdash b \rightarrow a; b \in S_{\mathcal{L}}(n); \\ &\quad \|a\| \leq m\} \\ &\cup \{a \mid \mathbf{LPR} \vdash f_{\circ}(b_0 \dots b_n) \rightarrow a; \\ &\quad \{b_0, \dots, b_n, c\} \subseteq S_{\mathcal{L}}(n), \|a\| \leq m\} \end{aligned}$$

Theorem 22 (The emptiness problem for logic of pure residuation). *The language generated by a logic of pure residuation $\mathcal{L}(\Sigma, D, f)$ is nonempty if and only if D is an element of the limit of $S_{\mathcal{L}}$.*

Lemma 23 ([2]). *Assume a language of pure residuation $\mathcal{L}(\Sigma, D, f)$ is given. We can construct an equivalent context-free grammar as follows. First we define the set of auxiliary symbols \mathcal{W} :*

$$\mathcal{W} := \{a \in T \mid \|a\| \leq m\}$$

We take D as the starting symbol. The set \mathcal{R} of rules is defined as follows:

$$\begin{aligned} \mathcal{R} &:= \{b \Rightarrow t \mid t \in \Sigma \text{ and } b \in f(t)\} \\ &\cup \{a \Rightarrow b_0 \dots b_n \mid a, b_0, \dots, b_n \in \mathcal{W} \text{ and} \\ &\quad \mathbf{LPR} \vdash f_{\circ}(b_0 \dots b_n) \rightarrow a\} \\ &\cup \{a \Rightarrow b \mid a, b \in \mathcal{W} \text{ and } \mathbf{LPR} \vdash b \rightarrow a\} \end{aligned}$$

Proof of theorem 22. This rest of the proof runs along the same line as the proof for **NL**. \square

5 Discussion

In this paper I presented the associative and nonassociative Lambek calculus and logic of pure residuation. My contribution consisted of a general way of solving the emptiness problem for those languages, without the need of translating to context-free grammars first. The proof of this solution made use of the conversion of the grammar to context-free grammar, and a solution for the emptiness problem for context-free grammar.

The use of investigating formal properties like the one presented in this paper is twofold. First, we can use results from formal language theory to obtain more knowledge about natural language. For example, if we assume Lambek calculus is a good representation of natural language, we can conclude that the emptiness problem is solvable for natural languages.

Secondly, we can also start with thinking about which properties are desirable in natural language, and look for a formalism which meets those desired properties. For example, if one believes that natural language should be modelled in a formalism which allows for a decidable emptiness problem - a reasonable assumption, because it seems plausible that given a set of words, humans can always find out if is possible to make a sentence with them - we know that Lambek calculus might be a good formalism, and we certainly know that language formalisms with an undecidable emptiness problem, like context-sensitive grammar, are not.

The method presented in this paper easily leads to a computational explosion. It might be an interesting future project to try optimizing the computational and space complexity of the solution. Other interesting future work might be trying to solve the emptiness problem for different extensions of Lambek calculus, like Lambek-Grishin calculus [7].

Acknowledgements

This paper was written as part of my master thesis project. I would like to thank Michael Moortgat and Vincent van Oostrom for supervising this project.

References

- [1] C. H. P. Harry R. Lewis. *Elements of the theory of computation*. Prentice-Hall, Inc., 1981.
- [2] G. Jäger. Residuation, structural rules and context freeness. *J. of Logic, Lang. and Inf.*, 13(1):47–59, 2004.
- [3] 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.
- [4] J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:363–386, 1958.
- [5] 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. American Mathematical Society, 1961.
- [6] M. Moortgat. Multimodal linguistic inference. *Journal of Logic, Language and Information*, 5:349–385, 1996.
- [7] M. Moortgat. *Symmetries in Natural Language Syntax and Semantics: The Lambek-Grishin Calculus*, pages 264–284. Springer Berlin / Heidelberg, 2007.
- [8] 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.