Proving Theorems of the Lambek calculus of order 2 in Polynomial Time

Aarts, H.M.F.M.

Published in:
Studia Logica

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Abstract. In the Lambek calculus of order 2 we allow only sequents in which the depth of nesting of implications is limited to 2. We prove that the decision problem of provability in the calculus can be solved in time polynomial in the length of the sequent. A normal form for proofs of second order sequents is defined. It is shown that for every proof there is a normal form proof with the same axioms. With this normal form we can give an algorithm that decides provability of sequents in polynomial time.

1. Introduction

The Lambek calculus was first presented in [4]. The Lambek calculus is a sequent calculus. A sequent expresses a consequence relation between the antecedent, a list of types (not a set), and a succedent type. Types are defined recursively:

- There is a set of basic types (e.g. \{s, np\}).
- If $A$ and $B$ are types, then $A/B$ and $A\setminus B$ are types.

We consider the (product-free) Lambek calculus, of which we give a sequent presentation in Figure 1. Greek capitals denote sequences of types. Empty antecedents are not allowed.

Rules that remove (or add) a connective in the antecedent are called left rules ([/L] and [\L]). In the right rules ([/R] and [\R]) a connective in the succedent is added or removed.

*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).
Axioms

\[ A \vdash A \quad (A \text{ basic}) \]

Left rules \hspace{2cm} Right rules

\[
\frac{\Gamma \vdash A \quad \Delta, B, \Delta' \vdash C}{\Delta, B/A, \Gamma, \Delta' \vdash C} \quad \text{/L} \hspace{2cm} \frac{\Gamma, A \vdash B}{\Gamma \vdash B/A} \quad \text{/R}
\]

\[
\frac{\Gamma \vdash A \quad \Delta, B, \Delta' \vdash C}{\Delta, \Gamma, A \backslash B, \Delta' \vdash C} \quad \text{/L} \hspace{2cm} \frac{A, \Gamma \vdash B}{\Gamma \vdash A \backslash B} \quad \text{/R}
\]

Figure 1: The Lambek calculus

Moortgat in [5] already showed that Lambek proofs are small, i.e. their size is linear in the number of connectives in a sequent. We can conclude from this that provability in the Lambek calculus is in the complexity class \( \mathcal{NP} \). All algorithms for theorem proving that have been developed earlier ([5], [3]) run in time exponential in the size of the sequent that must be proved. There are no polynomial algorithms known. In this paper we show a polynomial algorithm for a fragment of the Lambek calculus. This fragment is called the second order fragment.

Buszkowski [1] proved that the second order fragment is contextfree. There is no obvious way, however, to use this result in finding polynomial algorithms for the fragment. Buszkowski says in his proof [1, p. 149]: “Details of the construction, however, are pretty sophisticated (it would be interesting to estimate the required time which seems to be super-exponential)”. The grammar he gives is contextfree but exponential in size. Via Buszkowski’s construction it is not obvious how to obtain a polynomial algorithm for theorem proving in the second order fragment.

After description of the polynomial algorithm it is shown that, unfortunately, the method used in this paper can not be extended to larger fragments in a straightforward way.

The fragment can be defined as follows. We define the order of a type:

\[
\begin{align*}
\text{order}(A) &= 0 \text{ if } A \text{ is a basic type} \\
\text{order}(A/B) &= \max(\text{order}(A), \text{order}(B) + 1) \\
\text{order}(B \backslash A) &= \max(\text{order}(A), \text{order}(B) + 1)
\end{align*}
\]
Maybe it would be better to call this property of a type its depth but we call it order because others did so before. The order of a sequent is defined as the order of the highest order type occurring in it.

The fragment we restrict ourselves to in this paper is the fragment of sequents of order 0, 1 or 2. To give an example:

the sequent \( s/(np\ s), np\ (s/np), (s/np)\ s \vdash s \) has order 2.

The sequent \( s/(s/(np\ s)), np \vdash s \) has order 3 and is not in the fragment.

For convenience we only look at sequents whose succedent is a basic type\(^1\).

2. A normal form for proofs

Before we give the normal form we change the notation of types. This notation can be used for the Lambek calculus (no restriction on order). Because the Lambek calculus is associative the bracketing of a type is irrelevant to some extent.

**Axioms**

\[ A \vdash A \quad (A \text{ basic}) \]

**Left rules**

\[
\frac{\Gamma \vdash A \quad \Delta, (L \Rightarrow B \Leftarrow T), \Delta' \vdash C}{\Delta, (L \Rightarrow B \Leftarrow A, T), \Gamma, \Delta' \vdash C} \quad [/L]
\]

\[
\frac{\Delta, \Gamma, (T \Rightarrow B \Leftarrow L), \Delta' \vdash C}{\Delta, \Gamma, (T, A \Rightarrow B \Leftarrow L), \Delta' \vdash C} \quad [/L]
\]

**Right rules**

\[
\frac{\Gamma, \Delta, (L \Rightarrow B \Leftarrow T), A, \Delta' \vdash C}{\Gamma \vdash (L \Rightarrow B \Leftarrow A, T)} \quad [/R]
\]

\[
\frac{A, \Gamma, (T \Rightarrow B \Leftarrow L), \Delta' \vdash C}{\Gamma \vdash (T, A \Rightarrow B \Leftarrow L)} \quad [/R]
\]

A and B types, \( \Gamma, \Delta, \Delta', L \) and \( T \) lists of types.

Figure 2: The Lambek calculus in subcategorization list notation

We remove this unnecessary bracketing by giving two *subcategorization lists* to every type: one for the arguments that are expected at the left and one for the arguments that are expected at the right of some type. Following

\(^1\)We can easily transform sequents with a complex succedent to sequents with a basic succedent by application of right rules.
Buszkowski [1] a type \(((a \backslash (b \backslash c))/d)/e\) is converted into the type in subcategorization list notation \((b, a \Rightarrow c \Leftarrow e, d)\) where \(c\) is the final result, \(b, a\) is the list of arguments looked for on the left, \(e, d\) are the arguments on the right. Note that the subcategorization lists can be nested. We can define the Lambek calculus in this new notation as above (Fig. 2; \([\ ] \Rightarrow a \Leftarrow [\ ]\) equals \(a\)).

The system is equivalent to the Lambek calculus because the Lambek calculus is associative. The definition of order in this system is analogous to the definition we just gave.

In this new notation we can define a normal form as follows:

- the antecedents of the sub-sequents in the proof must lie on at most two intervals on the unfolded proof frame as defined in [7].

Roorda defines a mapping from types to trees in the (/, \) notation. This mapping is called unfolding.

First the labels "\(+\)" and "\(-\)" are assigned to the types. Types in the antecedent are labeled negative, in the succedent positive.

An initial tree for each type is defined. This tree has only one node. It is the type with its label. There are four rules to unfold nodes:

\[
\begin{align*}
\text{A} & \quad \text{B} & \quad \text{A} & \quad \text{B} & \quad \text{A} & \quad \text{B} \\
- & \quad + & \quad + & \quad - & \quad _ & \quad + \\
\text{A/B} & \quad \text{A/-B} & \quad \text{A/B} & \quad \text{A+B} & \quad \text{A/B} & \quad \text{A+B} \\
- & \quad - & \quad + & \quad + & \quad + & \quad + \\
\end{align*}
\]

Figure 3: Unfolding rules

If all leaves of the tree are unfolded, the unfolding is finished. If we unfold all types of a sequent, we get the proof frame of a sequent. Unfolding in the new notation is analogous to unfolding in the (/, \) notation. E.g. Figure 4 shows the proof frame of the sequent: \(np, np \backslash (s/np) \vdash s/np\) (or \(np, (np \Rightarrow s \Leftarrow np) \vdash s \Leftarrow np\)).
In the new notation we get similar proof frames, although the trees are flatter\(^2\) and not necessarily binary. A proof frame is a list of trees. We define an interval as a list of leaves that are adjacent in the proof frame. We define the last leaf of the last tree to be adjacent to the first leaf of the first tree. This means that a proof frame is circular instead of linear. Sometimes proof frames will be drawn circular in the rest of the paper.

We call the final conclusion in a proof the goal sequent. We can give proof frames for all sub-sequents in a proof in the same way as for the goal sequent. Because the subformula property holds in the system, the leaves in a proof frame of a sub-sequent form a subset of the leaves in the proof frame of the goal sequent. A very important property is that the leaves of any sub-sequent are in the same (circular) order as the leaves of the goal sequent. We can count the number of intervals the leaves of the antecedent of a sub-sequent are on with respect to the leaves of the proof frame of the goal sequent. Consider e.g. the goal sequent \(c \leftarrow d, e \leftarrow f, g, h \Rightarrow i, j \Rightarrow k \vdash n\). The antecedent of the sub-sequent \(c \leftarrow d, g, h \Rightarrow i \vdash n\) is on two intervals with respect to the goal sequent. The antecedent of the sub-sequent \(c \leftarrow d, g, j \Rightarrow k \vdash n\) is on three intervals. Figure 5 will make this clear.

\(^2\)The height of a tree is at most two because of the second order constraint.
The normal form for proofs is now defined as follows: all antecedents of sub-sequents in the proof must lie on at most two intervals. Therefore, a proof of the sequent \( c \Leftarrow d, e \Leftarrow f, g, h \Rightarrow i, j \Rightarrow k \vdash n \) containing the premise \( c \Leftarrow d, g, j \Rightarrow k \vdash n \) is not in normal form.

Definition of this normal form reduces the number of possible sub-sequents occurring in a proof enormously. There are exponentially many possible sub-sequents of a sequent (the set of leaves has exponentially many subsets). We restrict the number of possible sub-sequents (subsets) by admitting only sub-sequents whose antecedent is on at most two intervals.

If there are \( n \) leaves in the proof frame, there are fewer than \( \binom{n}{4} \) possibilities to choose two intervals (draw 4 borders out of \( n \) possible ones). The number \( \binom{n}{4} \) is smaller than \( n^4 \). If we have chosen two intervals and a succedent, there is at most one antecedent corresponding with these two intervals. We find the antecedent as follows. The unfolded proof frame shows us which types are subtypes from other types. The two intervals define a set of leaves. These leaves are (atomic) subtypes. In the set of types we replace any two types that form a bigger type (according to the proof frame) by this

---

Figure 5: Two vs. three intervals

---

\( c \Leftarrow d, g, h \Rightarrow i \vdash n \) 2 intervals  
\( c \Leftarrow d, g, j \Rightarrow k \vdash n \) 3 intervals

---

3In the list notation the notion subtype is a bit more complicated but this is not a real problem.
bigger type. Finally we should get a set of types with negative polarity that can not be combined further and we order these types according to their order in the proof frame. This gives us a circular ordering of the antecedent. The position of the succedent on the proof frame indicates the point where we have to "cut the circle". To give an example, remember the proof frame in Figure 4. The leaves are numbered in order to discriminate between them.

![Figure 6: A proof frame](image)

The interval $[3, 4]$ corresponds to the antecedent $s_3 \Leftarrow np_4$. The interval $[1, 2]$ does not give us an antecedent because $np_2$ has a positive polarity. The case where we have to solve the problem of turning a circular ordering into a linear one is the following. Suppose the goal sequent is $s_1 \Leftarrow (a_3 \Leftarrow c_2), a_4 \Leftarrow b_5, b_6 \Leftarrow c_7 \vdash s_8$ and we want to compute the antecedent corresponding to the pair of intervals $[2], [4, 7]$ when the succedent is $a_3$. First $\{c_2, a_4, b_5, b_6, c_7\}$ is turned into $\{c_2, a_4 \Leftarrow b_5, b_6 \Leftarrow c_7\}$. Because the succedent is $a_3$ the antecedent must be $a_4 \Leftarrow b_5, b_6 \Leftarrow c_7, c_2$. The sequent $c_2, a_4 \Leftarrow b_5, b_6 \Leftarrow c_7 \vdash a_3$ is not a sub-sequent because the order of the leaves is different from the order of the leaves of the goal sequent.

So a pair of two intervals uniquely defines an antecedent. Under the two intervals restriction the number of possible antecedents for some succedent is polynomial ($O(n^4)$).

Now we return to the normal form and show that no proofs are lost.

- the antecedents of the sub-sequents in the proof must lie on at most two intervals on the unfolded proof frame as defined in [7].

Suppose we have a proof that is not in normal form. Then we have to show that there is an equivalent proof in normal form. With an equivalent proof we mean a proof with the same axioms.
Let us first introduce the new system S of which we show that it is
equivalent with the Lambek calculus:

\[
\Delta_i \vdash A_i \quad \Gamma_j \vdash B_j \\
\Delta_1, \ldots, \Delta_k, (A_1, \ldots, A_k \Rightarrow D \Leftarrow B_1, \ldots, B_n), \Gamma_1, \ldots, \Gamma_n \vdash ([1] \Rightarrow D \Leftarrow []) \quad [\text{HNF}]
\]

\[
\Delta_i \vdash A_i \quad (A_1, \ldots, A_{k-1} \Rightarrow D \Leftarrow \Gamma'), \Gamma \vdash (\neg \Rightarrow \neg \Leftarrow L) \\
\Delta_{k+1}, \ldots, \Delta_n, (A_1, \ldots, A_k, \ldots, A_n \Rightarrow D \Leftarrow \Gamma'), \Gamma \vdash (\neg, A_k \Rightarrow E \Leftarrow L) \quad [\text{\[R\]}]
\]

\[
\Gamma_i \vdash A_i \quad \Delta, (\Delta' \Rightarrow D \Leftarrow A_{k+1}, \ldots, A_n) \vdash (\neg \Rightarrow \neg \Leftarrow T) \\
\Delta, (\Delta' \Rightarrow D \Leftarrow A_1, \ldots, A_k, \ldots, A_n), \Gamma_1, \ldots, \Gamma_{k-1} \vdash (\neg \Rightarrow E \Leftarrow A_k, T) \quad [\text{\[/R\]}]
\]

List of the form \(A, \ldots, B\) may be empty. Lists like \(A, \ldots, B, \ldots, C\) con-
tain at least \(B\).

**Equivalence of S and L**

**Proof of \(S \subseteq L\).** We prove this with induction to the length of proofs. Suppose we have L-proofs of all premises of \([\text{HNF}]\). Then you can apply the
left rules of L until you have an L-proof of the conclusion.

The same for the rule \([\text{\[R\}]}\]: apply \([\text{\[L\}]}\) rules of L on the premises and
apply the \([\text{\[/R\}]}\) rule of L. You have an L-proof of the conclusion.

The proof for \([\text{\[/R\}]}\) goes analogous.

**Proof of \(L \subseteq S\).** Define \(L'\) as the system with \([\text{HNF}]\) plus the right
rules of L: \([\text{\[/R\}]}\) and \([\text{\[R\}]}\). \(L'\) is equivalent with L [1]. So we have to prove
\(L' \subseteq S\). Before we prove this, we first prove a lemma about S.

**Lemma 1.** If \(\Gamma, B \vdash E\) with \(B\) atomic is provable in \(S\) then

- \(\Gamma\) is empty and \(E = B\) or
- \(\Gamma\) is of the form \(\Delta, (\Delta' \Rightarrow D \Leftarrow C_1, \ldots, C_m, B, \Delta''), \Theta_1, \ldots, \Theta_m\) and
the following sequents are provable:

\[- \Delta, (\Delta' \Rightarrow D \Leftarrow \Delta'') \vdash E \quad \text{and} \quad \Theta_i \vdash C_i\]

We prove this by induction on the length of the proof. Lemma 1 obviously
holds for axioms.
If the last rule used in the proof of $\Gamma, B \vdash E$ is HNF then $\Gamma, B \vdash E$ is not an axiom. We have to prove that it is an instance of the second case. We have

\[ \Delta_i \vdash A_i \quad \Gamma_j \vdash B_j \quad (j < n) \quad \Gamma_n, B \vdash B_n \]

\[
\Delta_1, \ldots, \Delta_k, (A_1, \ldots, A_k \Rightarrow E \Leftarrow B_1, \ldots, B_n), \Gamma_1, \ldots, \Gamma_n, B \vdash ([[ \Rightarrow E \Leftarrow ]]) \quad [\text{HNF}]
\]

Lemma 1 holds for the premise $\Gamma_n, B \vdash B_n$ via the induction hypothesis. If $\Gamma_n, B \vdash B_n$ is an axiom, Lemma 1 holds for the conclusion $\Gamma, B \vdash E$ because we can substitute in Lemma 1 $\Delta = \Delta_1, \ldots, \Delta_k, \Delta' = A_1, \ldots, A_k, D = E, m = n - 1, C_i = B_i, B_n = B, \Delta''$ and $\Gamma_n$ are empty and $\Theta_i = \Gamma_i$.

The following sequents have to be provable:

- $\Delta_1, \ldots, \Delta_k, (A_1, \ldots, A_k \Rightarrow E \Leftarrow [ ]) \vdash E$. Can be derived from $\Delta_i \vdash A_i$ with HNF.
- $\Gamma_i \vdash B_i$. We have the proofs.

If $\Gamma_n, B \vdash B_n$ is not an axiom and Lemma 1 holds for $\Gamma_n, B \vdash B_n$, then the last applied [HNF] rule has the following form:

\[ \Delta_i \vdash A_i \quad \Gamma_j \vdash B_j \quad \Delta', (\Delta'' \Rightarrow D \Leftarrow C_1, \ldots, C_m, B, \Delta'''), \Theta_1, \ldots, \Theta_m, B \vdash B_n \]

\[ \Delta_1, \ldots, \Delta_k, (\ldots \Rightarrow E \Leftarrow B_n), \Gamma_1, \ldots, \Gamma_n-1, \Delta', (\Delta'' \Rightarrow D \Leftarrow C_1, \ldots, C_m, B, \Delta'''), \Theta_1, \ldots, \Theta_m, B \vdash E \]

We have to prove Lemma 1 for the conclusion. The conclusion has the right form $(\Delta = \Delta_1, \ldots, \Delta_k, (\ldots \Rightarrow E \Leftarrow B_n), \Gamma_1, \ldots, \Gamma_n-1, \Delta')$. From Lemma 1 applied on the premise follows that the following sequents can be proved.

- $\Delta', (\Delta'' \Rightarrow D \Leftarrow \Delta'''') \vdash B_n$
- $\Theta_i \vdash C_i$

and we have to show that the following sequents can be proved (in order to prove Lemma 1 for the conclusion):

- $\Delta_1, \ldots, \Delta_k, (\ldots \Rightarrow E \Leftarrow \ldots, B_n), \Gamma_1, \ldots, \Gamma_n-1, \Delta', (\Delta'' \Rightarrow D \Leftarrow \Delta'''') \vdash E$

  Can be derived from $\Delta_i \vdash A_i, \Gamma_j \vdash B_j$ and $\Delta', (\Delta'' \Rightarrow D \Leftarrow \Delta'''') \vdash B_n$ with HNF.
- $\Theta_i \vdash C_i$
This completes the proof for the case that HNF was the last rule that was applied in the S-proof of \( \Gamma, B \vdash E \).

Suppose the last rule was \([\mathcal{R}]\). Then Lemma 1 holds for the premise
\[
(A_1, \ldots, A_k-1 \Rightarrow D \equiv \Gamma'), \Gamma, B \vdash (T \Rightarrow E \equiv L).
\]
This can not be an axiom because there are at least two types in the antecedent.

So it is not an axiom and Lemma 1 holds. We can choose \(\Delta\) in Lemma 1 empty or not empty. If \(\Delta\) is not empty the string \(\Gamma\) contains the type with the B-argument (and B is not in \(\Gamma'\)). Lemma 1 for the conclusion follows almost immediately from the fact that Lemma 1 holds for the premise like in the proof of the HNF case. The only thing that is different in Lemma 1 for the premise and the conclusion is the choice of \(\Delta\). The only thing we have to prove is that
from
\[
(A_1, \ldots, A_k-1 \Rightarrow D \equiv \Gamma'), \emptyset \vdash (T \Rightarrow E \equiv L)
\]
follows
\[
\Delta_{k+1}, \ldots, \Delta_n, (A_1, \ldots, A_k, \ldots, A_n \Rightarrow D \equiv \Gamma'), \emptyset \vdash (T, A_k \Rightarrow E \equiv L)
\]
This follows immediately if we apply \([\mathcal{R}]\) on \(\Delta_i \vdash A_i\) and
\[
(A_1, \ldots, A_k-1 \Rightarrow D \equiv \Gamma'), \Gamma \vdash (T \Rightarrow E \equiv L)
\]
If we choose \(\Delta\) empty in Lemma 1 then
\[
(A_1, \ldots, A_k-1 \Rightarrow D \equiv \Gamma') is (\Delta' \Rightarrow D \equiv C_1, \ldots, C_m, B, \Delta'')
\]
From Lemma 1 for the premise we know that

- \((A_1, \ldots, A_k-1 \Rightarrow E \equiv \Delta'') \vdash (T \Rightarrow E \equiv L)\)
- \(\Theta_i \vdash C_i\)

We have to prove for the conclusion:

- \(\Delta_{k+1}, \ldots, \Delta_n, (A_1, \ldots, A_n \Rightarrow E \equiv \Delta'') \vdash (T, A_k \Rightarrow E \equiv L)\). Apply \([\mathcal{R}]\).
- \(\Theta_i \vdash C_i\)

The third and last case is the case where the last rule applied in the proof of \(\Gamma, B \vdash E\) was \([\mathcal{R}]\).

The relevant premise is \(\Gamma_{k-1}, B \vdash A_{k-1}\). If it is an axiom we can match:
\[
C_1, \ldots, C_m = A_1, \ldots A_{k-2}, \Delta'' = A_k, \ldots, A_n, A_{k-1} = B, \Theta_1, \ldots, \Theta_m = \Gamma_1, \ldots, \Gamma_{k-2}
\]
We have to show:
• $\Delta, (\Delta' \Rightarrow D \Leftarrow A_k, \ldots, A_n) \vdash (L \Rightarrow E \Leftarrow A_k, T)$. Apply $[\text{IR}]$ on \\
$\Delta, (\Delta' \Rightarrow D \Leftarrow A_{k+1}, \ldots, A_n) \vdash (L \Rightarrow E \Leftarrow T)$ only.

• $\Gamma_i \vdash A_i$

If it is not an axiom we only have to prove that from

$$\Delta, (\Delta' \Rightarrow F \Leftarrow \Delta'') \vdash \Delta_{k-1}$$

follows

$$\Delta'', (\Delta' \Rightarrow D \Leftarrow A_1, \ldots, A_k, \ldots, A_n), \Gamma_1, \ldots, \Gamma_{k-2}, \Delta, (\Delta' \Rightarrow F \Leftarrow \Delta'') \vdash (L \Rightarrow E \Leftarrow A_k, T)$$

Apply $[\text{IR}]$ on $\Delta, (\Delta' \Rightarrow F \Leftarrow \Delta'') \vdash \Delta_{k-1}, \Gamma_i \vdash A_i$ and

$$\Delta'', (\Delta' \Rightarrow D \Leftarrow A_{k+1}, \ldots, A_n) \vdash (L \Rightarrow E \Leftarrow T).$$

This completes the proof of Lemma 1.

Lemma 2. As Lemma 1 but mirrored.

Now we return to the equivalence of $L'$ and $S$. We were proving that $L' \subseteq S$. $L'$ contains the following rule:

$$\Gamma, B \vdash (L \Rightarrow A \Leftarrow R) \quad [\text{IR}]_{L'}$$

We have to prove now that if we have an $S$-proof of the premise, we also have an $S$-proof of the conclusion.

We know $B$ is atomic (we are in the second order fragment). Lemma 1 holds. Therefore either $\Gamma, B \vdash (L \Rightarrow A \Leftarrow R)$ is an axiom or it equals

$$\Delta, (\Delta' \Rightarrow D \Leftarrow C_1, \ldots, C_m, B, \Delta'''), \Theta_1, \ldots, \Theta_m, B \vdash (L \Rightarrow A \Leftarrow R)$$

In the first case we can not apply the right rule. In the second case, apply $[\text{IR}]$ of $S$ on $\Theta_i \vdash C_i$ and

$$\Delta, (\Delta' \Rightarrow D \Leftarrow \Delta'') \vdash (L \Rightarrow A \Leftarrow R).$$

This gives the conclusion.

$$\Theta_i \vdash C_i, \Delta, (\Delta' \Rightarrow D \Leftarrow \Delta'') \vdash (L \Rightarrow A \Leftarrow R) \quad [\text{IR}]_{S}$$

For $[\text{IR}]$ the proof is similar, using Lemma 2.

We know from [1] that $L' = L$ and we have proved $L' \subseteq S$ and $S \subseteq L$. So $S$ is equivalent with $L$.

Now we can explain why we introduced the system. The antecedents of all premises in an $S$-proof are on one interval! The antecedent of the goal
sequent is on one interval by definition. If the antecedent of the conclusion is on one interval, so are the antecedents of the premises. Therefore the antecedents of all premises are on one interval.

We can "simulate" the S-proof with an L-proof as we did in proving that \( S \subseteq L \). When we are in a HNF part of the S-proof we can choose between the L-rules \([/L]\) and \([\langle L\rangle]\). Assume that we first apply all \([/L]\) and then all \([\langle L\rangle]\) rules. In a \([/R]\) part of the S-proof the \([/R]\) of L is as high as possible in the proof. Now we have exactly one L-proof for every S-proof. In the simulation of the HNF part of the S-proof the antecedent always is on two intervals. In the simulation of the \([/R]\) and \([\langle R\rangle]\) part of the S-proof the antecedent always is on two intervals.

So if we restrict ourselves in the system L to premises whose antecedent is on two intervals we will still find all proofs (when we are in the second order fragment). So the normal form is correct.

3. The algorithm

A simple algorithm to find out whether or not there is a proof for some theorem in the system L with subcategorization list notation is the following.

- If the theorem is an axiom we have a proof.
- If the theorem is not an axiom
  
  - we try to match it against the conclusion of a right rule. We have one premise now that has to be proved.
  
  - or we try to match it against the conclusion of a left rule. We have two premises now that have to be proved.

We call this method backward chaining of the logical rules. The nondeterminism can be eliminated by trying all possibilities in a depth–first or breadth–first manner. The rules show that the number of connectives will decrease with every backward application of a logical rule. The connectives are removed one by one. This guarantees that the algorithm will stop. The search space is finite because the number of connectives decreases with every application of a rule.

This is the standard algorithm for Lambek theorem proving. It shows that theorem proving in Lambek calculus is decidable. If we use the subcategorization list notation and allow only premises whose antecedents are on two intervals we almost have a polynomial algorithm. The only thing we have to add is memoization [2, pp. 312-314]. Memoization means that we put in a table the result of our first attempt to compute the solution of a
problem. If we later want to solve this problem again we can see the answer immediately in the table.

In this algorithm we store the result of any attempt to prove a sub-sequent in a table. When we try to prove the same sub-sequent again we know the answer immediately because it is in the table.

The complexity of the algorithm is as follows. Take for $n$ the number of connectives in a sequent. We have to compute $O(n^6)$ times whether a sequent is provable or not ($O(n^4)$ choices for the antecedent and $O(n^2)$ for the succedent). In a computation of validity of a sub-sequent we have to prove at most $O(n^2)$ pairs of premises. In the left rules we have to decide which connective must be removed, or better which element is removed from the subcategorization list. We have $O(n)$ possibilities. After this the choice of $\Gamma$ in the left rule can be made in $O(n)$ different ways.

Table look-up can be done in constant time. In time $O(n^2)$ we can compute (or look up) all subgoals of a goal. We have to do this $O(n^6)$ times. The total complexity of the algorithm is $O(n^8)$.

This algorithm is an example of the technique of *dynamic programming* [2, Chapter 16]. Another possible algorithm would be to start with the axioms and reason from the axioms to the conclusion. If the sub-conclusions are constrained so that their antecedents are on two intervals we can obtain a polynomial algorithm too. This is a general property of algorithms based on dynamic programming: they can be reversed easily in general.

4. Discussion

We have given a polynomial algorithm for deciding provability in the second order fragment of Lambek calculus. The most important idea is the normal form which restricts possible sub-sequents to those whose antecedents are on two intervals (or less). Once we have the normal form, the polynomial algorithm is easy to find.

The algorithm can also be used for the Lambek calculus with empty antecedents. We do not have to change anything. We only have to stipulate that empty antecedents are on one interval.

We can show that the condition of three intervals does not work in higher order fragments ($> 2$). The type $(((c \Rightarrow c) \Rightarrow b) \Rightarrow b)$ has order 3. Consider the sequent:

$$(c \Rightarrow b), ((c \Rightarrow b) \Rightarrow b), (((c \Rightarrow c) \Rightarrow b) \Rightarrow b) \vdash b$$

A necessary subproof is:
\[ c, (c \Rightarrow c), (c \Rightarrow b) \vdash b \]

The antecedent of this sub-sequent is on 3 intervals. By repeating the type \(((c \Rightarrow c) \Rightarrow b) \Rightarrow b\) we can construct a counterexample to any fixed bound on the number of intervals one needs in a proof.

In order to find a polynomial algorithm for the whole calculus (so not for the second order fragment only) we have to do something different. It is worth noting that the cut rule is an admissible rule in the Lambek calculus. Algorithms which make use of the cut rule have gained interest since Pentus has proved that the Lambek calculus is context free [6].

We have given an \(O(n^8)\) algorithm. Maybe it is possible to improve the algorithm such that the bound is decreased. The main goal in this paper however was to show a polynomial time algorithm.

Acknowledgements

Leen Torenvliet and Marten Trautwein read earlier drafts of this paper. Their comments are much appreciated. Wojciech Buszkowski helped me to improve some difficult proofs a lot. His suggestions were very valuable.

References


