Proof Tables for the Lambek Calculus

Dirk Heylen
Research Institute for Language and Speech
Trans 10
NL-3512 JK Utrecht
E-mail: heylen@let.ruu.nl

Abstract

Based on the initial suggestion from Lambek, a number of people have implemented the Parsing as Deduction program by using the Gentzen proof method as the decision procedure for the Lambek calculus. Within proof theory there are, however, a number of alternative proof systems related to that of Gentzen (from semantic tableaux to resolution). In trying to adapt these alternative procedures to the categorial calculus, a number of interesting results have been obtained. Some of these are discussed in this paper. We start off by briefly introducing the Lambek calculus and some of the alternative proof procedures. Next we discuss two results that have emerged in trying to use these other systems as decision procedures for the Lambek calculus. We then point out how they can be made profitable in terms of parsing algorithms.

1 Introduction

As early as 1958, [Lambek, 1958] introduced the concept of parsing as deduction, by adopting the Gentzen sequent proof system, as a decision procedure for his categorial calculus. Renewed interest in categorial grammars (in particular in the field of computational linguistics) has led to a rediscovery of this work (see [van Benthem, 1986, Moortgat, 1988], for details). Several researchers have suggested various ways to improve on the efficiency of the use of the Gentzen proof procedure as a parsing algorithm. Grosso modo we can distinguish three approaches:

- the use of heuristics and invariants [van Benthem, 1986, Moortgat, 1988, van der Wouden and Heylen, 1988];
- the normalization of proofs [Hepple, 1990, König, 1989, Moortgat, 1990];
- the use of alternative proof procedures [Roorda, 1990].

The search space defined by the Gentzen proof procedure for the Lambek calculus can be reduced significantly by using heuristics and invariants. Certain heuristics may rank the paths in the search space into some order that can guide a best-first algorithm. Although they need not guarantee any optimization in some worst-case instances, they may significantly reduce the processing costs for the average case[1]. As far as invariants are concerned, we will refer to the count invariant later on in this paper.

[1]See [Moortgat, 1988, p. 151 ff.] for an example.

Normal forms can be exploited to reduce the size of the search space as well. It seems therefore worthwile to investigate how the results obtained in the field of Automatic Theorem Proving can improve the parsing procedure (given that we have defined it as a proof procedure itself).

The exploration of normal forms may soon lead to the adaptation of other proof procedures. This paper discusses some of the results obtained during this process of adaptation. In [Roorda, 1990] the reader can find another example.

2 Categorial Grammar

2.1 Different versions of Categorial Grammar

Application The best known rule of categorial grammar is without any doubt that of functional application which forms the basis of the Adjukiewicz Bar-Hillel system. In a sequent notation we can write this rule as follows.

$$\frac{\mathrm{U,X,V} \rightarrow \mathrm{Z} \qquad \mathrm{T} \rightarrow \mathrm{Y}}{\mathrm{U,X/Y,T,V} \rightarrow \mathrm{Z}}\ [L/]$$

Here, U, T, and V are sequences of categories with T non-empty. X, Y and Z are categories. Categories are defined inductively to be the smallest set (Cat) such that:

X ∈ Cat if X ∈ BasCat (a finite, non-empty set of atoms)
X/Y ∈ Cat if X ∈ Cat and Y ∈ Cat
Y\X ∈ Cat if X ∈ Cat and Y ∈ Cat

A category is the name of a set of expressions. The interpretation of complex categories is defined as:

$\mathrm{X/Y} = \{z \in V^+ \mid \forall y \in Y : zy \in X\}$
$\mathrm{Y\backslash X} = \{z \in V^+ \mid \forall y \in Y : yz \in X\}$
where V is the vocabulary of the language.

A sequent of the form $C_1, ...C_n \rightarrow Z$ should be read "the set of expressions which are the concatenations of expressions of categories $C_1, ..., C_n$ are a subset of the expressions of category Z". The arrow separates the antecent from the succedent part of a sequent.

In $[L/]$ the sequents above the line are called the premises, whereas the sequent under it is called the conclusion. One should read this rule as "the sequence of categories U,X/Y,T,V reduces to the category Z if there is a subsequence T to the right of X/Y which reduces to Y and the sequence U,X,V reduces to Z". The corresponding rule for Y\X is labeled $[L\backslash]$.

$$\frac{\mathrm{T} \rightarrow \mathrm{Y} \qquad \mathrm{U,X,V} \rightarrow \mathrm{Z}}{\mathrm{U,T,Y\backslash X,V} \rightarrow \mathrm{Z}}\ [L\backslash]$$

Abstraction The [L/] and [L\] rules remove slashes from the antecedent part of a sequent, if we read the rules in a backward chaining fashion (from conclusion to premises). [R/] and [R\] remove slashes from the succedent part of a sequent.

$$\frac{\mathrm{T,Y} \rightarrow \mathrm{X}}{\mathrm{T} \rightarrow \mathrm{X/Y}}\ [R/]$$

$$\frac{\mathrm{Y,T} \to \mathrm{X}}{\mathrm{T} \to \mathrm{Y\backslash X}}\ [R\backslash]$$

The interpretation of [R/] is "a sequence of expressions T is a subset of the set of incomplete expressions X/Y if the concatenation of this sequence with an expression of category Y yields an expression of category X".

Axiom The Lambek calculus is defined by adding the following axiom schema to the rules of inference $([L\backslash], [L/], [R\backslash], [R/])$.

$$\mathrm{X} \to \mathrm{X}\ [Axiom]$$

Example Parsing a sentence $w_1, \ldots, w_n$ can be thought of as constructing a proof tree for the sequent $T(w_1), \ldots, T(w_n) \to S$, where $T(w_i)$ is a category assigned to the word w_i by the lexicon and S is the category of expressions called sentence. A proof tree is a tree with the initial sequent as the root, the nodes instances of rules of inference and the leaves instances of axioms.

Supposing that the lexicon specifies the following for the words in T: { (the,NP/N), (donkey,N), (farmer,N), (ate,(NP\ S)/NP) }, we can prove that the expression 'The farmer ate the donkey' is a sentence as follows.

'The farmer ate the donkey'

$$\dfrac{\dfrac{\dfrac{\mathrm{NP} \to \mathrm{NP} \qquad \mathrm{N} \to \mathrm{N}}{\mathrm{NP/N, N} \to \mathrm{NP}} \qquad \mathrm{S} \to \mathrm{S}}{\mathrm{NP/N, N, NP\backslash S} \to \mathrm{S}} \qquad \dfrac{\mathrm{NP} \to \mathrm{NP} \qquad \mathrm{N} \to \mathrm{N}}{\mathrm{NP/N, N} \to \mathrm{NP}}}{(\mathrm{NP/N})_U,\ \mathrm{N}_U,\ (\mathrm{NP\backslash S})_X/\mathrm{NP}_Y,\ (\mathrm{NP/N})_T,\ \mathrm{N}_T \to \mathrm{S}_Z}$$

The easiest way to read this proof tree is from the bottom upwards. The root of the tree is the sequent whose antecedent consists of the categories corresponding to the words of the sentence, and whose succedent contains only S. The subscripts are there only for ease of reference. They indicate how this part of the tree maps onto the [L/] rule: X = NP\S; Y = NP; Z = S; U = NP/N, N; T = NP/N, N; V = $\emptyset$. The reader should now be able to figure out how the rest of the proof tree was constructed. For more details he can consult the relevant literature [van Benthem, 1986, Lambek, 1958, Moortgat, 1988].

2.2 Parsing as Theorem Proving

The [L] and [R] rules mimic the logical modus ponens and conditionalization rules from natural deduction respectively. The slashes correspond to the implication sign of propositional logic. As this may not be immediately obvious to the reader who is unfamiliar with the Gentzen proof procedure we show the corresponding rule for propositional logic. The Gentzen rule for the corresponding implication goes as follows[2]:

$$\frac{\Gamma \to B, \Delta \qquad \Gamma, A \to \Delta}{\Gamma, (B \Rightarrow A) \to \Delta}\ [\Rightarrow: L]$$

[2]Γ and Δ are sets of propositions, A and B propositions.

The antecedent contains the propositions one tries to make true, the succedent the propositions one wants to make false. The Gentzen rule [$\Rightarrow$: L] should now be obvious: if one has proven that B is false, or that A is true, one can conclude the truth of $(B \Rightarrow A)$. Proofs are ad absurdum. To prove the validity of a formula one tries to 'falsify' it. Thus one starts the proof tree by putting the formula in the succedent part of a sequent. Next one tries to show (by applying the rules of inference) that each valuation (assignment of truth values to the atomic propositions in the formula) leads to a contradiction. This is the case when all the leaves of a finished proof tree[3] are axioms. These are of the form:

$$\Gamma, P \rightarrow \Delta, P$$

Clearly such sequents are contradictions as they can only be made true if P is both assigned true and false.

Example The following tree proves the formula $((p \Rightarrow q) \wedge p) \Rightarrow q$. It makes use of the rules [$\Rightarrow$: R] and [$\wedge : L$]. We introduce these first.

$$\frac{\Gamma, A, B \rightarrow \Delta}{\Gamma, (A \wedge B) \rightarrow \Delta}\ [\wedge : L]$$

$$\frac{\Gamma, A \rightarrow \Delta, B}{\Gamma \rightarrow \Delta, (B \Rightarrow A)}\ [\Rightarrow : R]$$

$$\dfrac{\dfrac{\dfrac{p \rightarrow p, q \qquad\qquad p, q \rightarrow q}{(p \Rightarrow q), p \rightarrow q}}{(p \Rightarrow q) \wedge p \rightarrow q}}{\rightarrow (((p \Rightarrow q) \wedge p) \Rightarrow q)}$$

2.3 Gentzen Conjunctive Normal Form and Resolution

The resolution method for propositional logic rests on the following tautology:

$$((A \vee P) \wedge (B \vee \neg P)) \Leftrightarrow ((A \vee P) \wedge (B \vee \neg P) \wedge (A \vee B))$$

The equivalence states that we can add the clause $(A \vee B)$ to a formula (which is in conjunctive normal form[4]) if A and B are members of two different clauses one of which contains some atomic proposition, and the other its negation. The clause $(A \vee B)$ is called the *resolvent* of the two clauses. The resolution proof starts with negating the formula one wants to prove (i.e. the proof is ad absurdum) and transforming this into clausal form. The proof proceeds by adding resolvents to the formula until either no more resolvents can be added or the formula contains clauses of the form P and $\neg P$ (clearly a contradiction)[5]. In the first case the proof fails, whereas in the second the original formula comes out as a theorem.

[3] A proof tree is finished when each leaf is either an axiom or it has only atomic propositions.

[4] Remember that a proposition A is in conjunctive normal form if it is a conjunction of disjunctions of literals. Each conjunct is called a clause.

[5] An alternative way of stating the second case is by saying that the *empty clause* is derived. Clearly, the empty clause is the resolvent of the clauses P and $\neg P$.

Example The resolution proof of the formula $(((p \Rightarrow q) \wedge p) \Rightarrow q)$ proceeds as follows:

1. $((p \Rightarrow q) \wedge p) \Rightarrow q$ (Theorem)
2. $\neg(((p \Rightarrow q) \wedge p) \Rightarrow q)$ (Negation of Theorem)
3. $(\neg p \vee q) \wedge p \wedge \neg q$ (Clausal form of Negated Theorem)
4. $(\neg p \vee q) \wedge p \wedge \neg q \wedge \neg p$ (Addition of the resolvent $\neg p$ from the clauses $(\neg p \vee q)$ and $\neg q$)

The last formula is a contradiction as it contains both the clauses p and $\neg p$. Since this formula is logically equivalent to the second formula (the negation of the theorem) this shows that the original formula is indeed valid (i.e. a theorem).

Gallier [Gallier, 1987] introduces the resolution method by first defining a special variant of the Gentzen proof system and then showing how each proof in this GCNF'-system (for Gentzen Conjunctive Normal Form) can be turned into a resolution proof and vice versa. The obvious question to ask is whether the same can be done for the Lambek calculus. As we have shown, for the resolution method to work, the proposition to which it is applied should be in conjunctive normal form. One of the main difficulties in converting these proof systems for propositional logic into ones for categorial calculi has been to define something which can rightfully be called the conjunctive normal form of categories. To solve this problem one has to decompose the information hidden within a category.

In the next section we define a conjunctive normal form representation of categories. In the last section we show how this can be turned to profit even without using it to define GCNF' or resolution analogues for the Lambek calculus. After we have shown how the information contained in a sequent can be spread out in a table, we proceed by defining some constraints (invariants) that may help in defining alternative proof (or parse) procedures.

3 Decomposing Categories

3.1 Lambek: logic of occurrences

Given that axioms are of the form X $\rightarrow$ X (where we may safely restrict the categories to basic ones), we can expect that for each occurrence of a basic category there should be exactly one other matching occurrence of the same category as a necessary condition on provable sequents [van Benthem, 1986, p. 4].

We can formulate this condition as follows. We first define the count value of a basic category A in a category X.

$$count(A, X) = 1 \Leftarrow (X = A).$$
$$count(A, X) = 0 \Leftarrow (X \in BasCat) \wedge (X \neq A).$$
$$count(A, X) = count(A, B) - count(A, C) \Leftarrow (X = B/C) \vee (X = C\backslash B).$$

The count value of a basic category A in a sequence of categories $X_1, \ldots, X_n$ is defined as follows.

$$count(A, \langle X_1, \ldots, X_n \rangle) = count(A, X_1) + \ldots + count(A, X_n)$$

Given these definitions we can formulate what has become known as the count invariant (first introduced by van Benthem).

Count invariant
There is no proof for the sequent $T \rightarrow C$ if there is a basic category A for which $count(A, T)$ is different from $count(A, C)$.

The count procedure relies on the fact that 1) in a category such as X\Y (or Y/X) X appears in a negative position and Y in a positive one; 2) in an axiom positive and negative occurrences match. This idea of positive and negative occurrences gives us the basic insight into how to get to the conjunctive normal form of a category. In turning a proposition $P \Rightarrow Q$ into conjunctive normal form we use the equivalence $(P \Rightarrow Q) \Leftrightarrow (\neg P \vee Q)$, to eliminate the implication sign. Something similar should be done for categories. We shall have to find a procedure that somehow removes the slashes by defining an equivalence between $A\backslash B$ and something like $\neg A \vee B$.

3.2 Signed categories

We can write the above equivalence in terms of signed formulae [Smullyan, 1968] as $\mathbf{T}(P \Rightarrow Q) \Leftrightarrow \mathbf{F}(P) \vee \mathbf{T}(Q)$. We will write the categorial equivalent as: $+(A\backslash B)\Leftrightarrow(\leftarrow(A), +(B))$. We will use the following set of signs: $\{\rightarrow, \leftarrow, +\}$. Intuitively, they have the following interpretation: + means "occurs positively" whereas both → and ← mean "occurs negatively". The arrows correspond to the directionality indicated by the slashes. If we do not distinguish between → and ←, essential information will be lost.

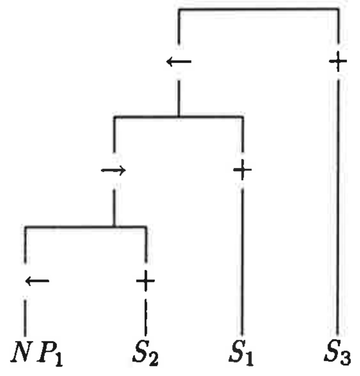
Instead of assigning one sign we will assign a list of signs to account for more complex categories such as S/(NP\S) or (S/(NP\S))\S. Whereas in propositional logic **FFA** reduces to **TA**, it is not clear how we should reduce →→ A, ←← A, →← A or ←→ A (without loss of information). Notice that this is an important problem (which we will not go into here). We present the procedure that assigns the appropriate list of signs to each basic category in terms of a Horn Clause program.

```
sign(X, ⟨[ ],X⟩) ⇐ basic-category(X).
sign(X\Y,Xs\Ys) ⇐ sign(X,Xa) & add(←,Xa,Xs) & sign-aux(Y,Ys).
sign(Y/X,Ys/Xs) ⇐ sign(X,Xa) & add(→,Xa,Xs) & sign-aux(Y,Ys).

sign-aux(X,⟨[+],X⟩) ⇐ basic-category(X).
sign-aux(X,Xs) ⇐ sign(X,Xs).

add(S,⟨L,X⟩,⟨[S|L],X⟩) ⇐ basic-category(X).
add(S,X\Y,Xs\Ys) ⇐ add(S,X,Xs) & add(S,Y,Ys).
add(S,Y/X,Ys/Xs) ⇐ add(S,X,Xs) & add(S,Y,Ys).
```

The sign of $(S_1/(NP_1\backslash S_2))\backslash S_3$, for example, is

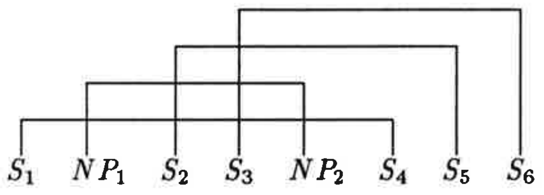
$$(\langle[\leftarrow, +]_{S1}\rangle/(\langle[\leftarrow, \rightarrow, \leftarrow]_{NP1}\rangle\backslash\langle[\leftarrow, \rightarrow, +]_{S2}\rangle))\backslash\langle[+]_{S3}\rangle$$

This can be easily deduced from the following tree:

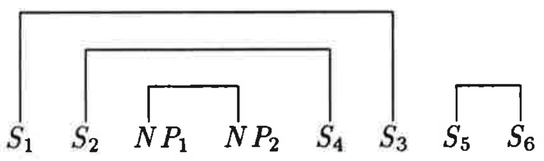


3.3 Directionality

Lambek [Lambek, 1987, p. 19] motivates his notation for categories X\Y and Y/X, where X is the domain of the functor and Y the range, by showing that in cases of simple application such as Y/X, X → Y and X, X\Y → Y, the X's are adjacent. The property no longer holds in the case of more complex categories. If we take the example of $S_1/(NP_1\backslash S_2)$ $(S_3/(NP_2\backslash S_4))\backslash S_5 \rightarrow S_6$ and connect the categories that turn up as pairs in axioms we get the following:



What we get are lines that cross each other. It is, however, relatively simple to define a procedure that will put the basic categories into the right order. The trick is to recursively reverse the order of categories which are in negative positions. The reason is obvious when one considers more closely the definition of the interpretation of categories as sets of expressions. As an example we order the categories in $(S_3/(NP_2\backslash S_4))\backslash S_5$. We start off by ordering $(NP_2\backslash S_4)$ as [NP_2,S_4]; the simple case. When ordering S_3 and [NP_2,S_4] we notice that $NP_2\backslash S_4$ was in negative position in the category $S_3/(NP_2\backslash S_4)$ so we will reverse this list to get: [S_3,[S_4,NP_2]]. When putting this result to the left of S_5 we again reorder the lists (recursively) to get [[[NP_2,S_4],S_3],S_5]. Ordering the categories of $S_1/(NP_1\backslash S_2)$ results in [S_1,[S_2,NP_1]]. If we now draw the lines we get:



We can write a category such as $(S/(NP\backslash S))\backslash S$ in the format:

$[[[\leftarrow\rightarrow\leftarrow NP, \leftarrow\rightarrow +S], \leftarrow +S], +S]$

If we signed complex categories as well, this would yield:

$$[\leftarrow [\rightarrow\leftarrow [\leftarrow\rightarrow\leftarrow NP, \leftarrow\rightarrow +S], \leftarrow +S], +S]$$

We leave it to the reader to construct an algorithm that reconstructs the traditional notation from this one.

4 Constraints on a Table

4.1 A simple table

Given the definitions of sign and ordering, we can represent a sequent in the form of a table:

	NP/N		N	(NP\S)/NP			NP/N		N	S
	NP	N		NP\S		NP	NP	N		
				NP	S					
N		→	+					→	+	
NP	+			←		→	+			
S					+					←

We put the succedent category in the table as well and signed it ←. We have also put the basic categories in the non-crossing order.

The attentive reader should be able to observe the following:

1. In each row (i.e. for each basic category) there are as many arrows as there are + signs (this corresponds to the negative/positive occurrences come in pairs claim).
2. For each → there is a + in the same row in a column that is to the right of the arrow and for each ← there is a + to its left.
3. If we connect the basic categories as we did in the previous sections we again get non-crossing lines.

4.2 A more complex one

The observations (1) and (2) are not the proper generalizations as the following table for $S/(NP\backslash S), (S/(NP\backslash S))\backslash S \rightarrow$ S shows.

	S/(NP\S)			(S/(NP\S))\S				S
	S	NP\S		S/(NP\S)			S	
		S	NP	NP\S		S		
				NP	S			
NP			→←	←→←				
S	+	→+			←→+	← +	+	←

Observation (2) told us to interpret an arrow as a directive to look for a plus sign in the same row and in the direction pointed to by the arrow. If we have more than one sign, ←→← for instance, we should interpret this as look to the left (←) for a category signed →←.

4.3 Parsing in terms of constraint propagation

Gentzen proof trees (for the Lambek calculus) have leaves labeled with axioms of the form $X \rightarrow X$. The application of the rules of inference and of the structural rules on a sequent determine which categories will ultimately be linked to each other. The search for a proof tree can be a time-consuming process, since many of the possible paths that can be traversed lead to blind alleys. The count-invariant tells us that we should not try to prove a sequent for which the count values for some category in the antecedent and the succedent are different. Given our new notation for categories, we can easily define additional invariants. This is what we will do next. Instead of constructively building proof trees (or finding the axioms) using the rules of inference, we will reverse the procedure and define constraints on the set of all pairs of basic categories occurring in a sequent. What we try to do is to find out as quickly as possible whether or not there is a set of axioms that proves the sequent. We use tables such as the ones above to represent sequents.

4.4 Conditions

We will refer to basic categories by means of their position in a table (we number columns from left to right starting from 1).

Suppose that P is the set of all pairs of column positions $\langle i, j\rangle$. We should now be able to find at least one subset of P, called $\mathcal{A}(P)$, that satisfies the following conditions.

1. $\langle i, j\rangle \in \mathcal{A}(P)$ only if $i < j$.
2. $\langle i, j\rangle \in \mathcal{A}(P)$ only if the categories at positions i and j are the same.
3. $\langle i, j\rangle \in \mathcal{A}(P)$ only if the sign of i matches the sign of j, where matching can be defined as follows.

   $match([+], [\leftarrow]).$
   $match([\rightarrow], [+]).$
   $match([\rightarrow | R], R).$
   $match(R, [\leftarrow | R]).$

   We call this the matching constraint. It is a more constraining version of the count invariant.
4. Each column position should occur only in one pair of $\mathcal{A}(P)$.
5. There should be no pairs $\langle i, n\rangle, \langle j, m\rangle$ in $\mathcal{A}(P)$ such that i < j < n < m. We call this the crossing constraint.

Notice that for ambiguous cases such as $N/N, N, N\backslash N \rightarrow N$, there will be two sets satisfying these conditions.

4.5 Comments

- We can interpret the conditions as constraints on tables. This makes it possible to reformulate the parsing procedure in terms of constraint propagation. Efficiency is obtained by using a notation or data structure that makes it easy to manipulate the essential information. Every vital piece of information is readily available, whereas with the unfolding of the proof tree this information may be computed spuriously.

- As far as proof theory is concerned, we should point out that the procedure or constraints defined are reminiscent of (or inspired by) the notion of Hintikka sets for propositional logic. The constraints are helpful as well in implementing other proof methods such as resolution as we will see in the next section.
- The table (as data structure) and the constraints (as procedures) help to solve some important problems in the parsing of categorial grammars. They can be put to use when normalizing proofs and they get rid of some cases of spurious ambiguity[6].

5 Proof Tables and Connection Graphs

The initial motivation for decomposing categories was to explore the possibility of adapting other proof procedures for the Lambek calculus than the Gentzen sequent proofs. In this section we will briefly review the connection graph procedure presented in [Kowalski, 1979] (a special form of the resolution method) and outline a possible relation with the constraints on the proof tables above.

5.1 Connection Graphs

Clauses The formulas that participate in the connection graph procedure are clauses of the form:

$$A_0 \wedge \ldots \wedge A_n \Rightarrow B_0 \vee \ldots \vee B_m$$
with $n, m \geq 0$

They are written as:

$$B_0, \ldots, B_m \leftarrow A_0, \ldots, A_n$$

The A formulas are the joint conditions and the B formulas the alternative conclusions. Horn clauses have the restriction $m \leq 1$. Notice that [Kowalski, 1979] deals with first-order formulas, whereas we will restrict ourselves to the case where the formulas are simple propositions.

Resolvent A resolvent of two clauses exists if there is a proposition P that occurs as a condition in the one and as a conclusion in the other. The conditions of the resolvent are all the conditions of both parent clauses minus P and its conclusions are the conclusions of both parent clauses again minus P.

Connection Graphs The basic data structure[7] that is used in the proof procedure is called a *connection graph.* Given a set of clauses, the initial connection graph is constructed by linking identical propositions that occur as conclusion and condition in different clauses. Associated with each link (arc) in the graph is the corresponding resolvent and each resolvent that can be generated from these clauses has its arc.

[6]Remember that in general spurious ambiguity refers to the situation when different derivations yield the same structure.

[7]Instead of sequents.

Proof Procedure Kowalski defines the proof procedure associated with such graphs as follows [Kowalski, 1979, p. 165]:

> The *connection graph proof procedure* begins with an initial connection graph and processes it repeatedly until the empty clause is generated. It *processes a connection graph by*
>
> 1. repeatedly deleting clauses containing unlinked atoms and deleting their associated links until all such clauses have been deleted and then
> 2. selecting a link, deleting it and adding the resolvent and its associated new links to the graph.

Example We illustrate the procedure by proving the formula $(((p \Rightarrow q) \wedge p) \Rightarrow q)$ again. The clausal form of its negation is given by the following set of formulas: $\{q_1 \leftarrow p_2, p_3 \leftarrow, \leftarrow q_4\}$[8]. The initial connection graph connects the following pairs $\langle q_1, q_4\rangle$ and $\langle p_2, p_3\rangle$. There are no unlinked atoms in this graph so we select a link, $\langle q_1, q_4\rangle$, delete it and add the resolvent with the new links. The result is a graph consisting of the set of clauses $\{\leftarrow p_2, p_3 \leftarrow\}$ and the arc $\langle p_2, p_3\rangle$. If we resolve on this arc we derive the *empty clause* which proves the validity of the original formula.

5.2 Connection Graphs for Categories

Categorial Graphs As we have pointed out above, the slashes in categories correspond to the implication sign in propositions. Given this correspondence we can rewrite the categories in Horn clause form[9]. Some examples,

N	$N \leftarrow$
NP/N	$NP \leftarrow N$
$(NP\backslash S)/NP$	$S \leftarrow NP \wedge NP$

The connection graph for the sentence 'The farmer ate the donkey' consists of the nodes labeled by the following categories: $\langle NP \leftarrow N, N \leftarrow, S \leftarrow NP \wedge NP, NP \leftarrow N, N \leftarrow\rangle$. Notice that there are two nodes labeled with $NP \leftarrow N$ and two nodes with N because we are dealing in the categorial case with a logic of occurrences. If we want to prove that these categories reduce to a sentence we have to add the node labeled $\leftarrow S$. We can now construct the initial connection graph by adding the links just as in the propositional case. Applying (literally) the same proof procedure on this graph as the one defined for the propositional variant, we find out that 'The farmer ate the donkey' is indeed a sentence.

5.3 Constraining connections

In the translation of categories into clauses essential information has been left out. The information on the directionality of the slash has disappeared and the graph does no longer contain information about the ordering of the categories (of the words in the sentence). The problem can be solved by redefining the construction algorithm for the initial graph. The categories are also assigned an index (the column position in a proof table) and their sign.

[8]The subscripts are there for ease of reference only.
[9]We have changed the comma back into the $\wedge$ symbol.

When adding links, the various constraints on pairings as defined in terms of proof tables should be taken into account. The resolution procedure for the categorial case is now a straightforward adaptation of the algorithm above.

Example The initial graph of the example is given by the following pair of sets (the first list the nodes, the second the links, we leave the signs unspecified):

$$\langle\{NP_1 \leftarrow N_2, N_3 \leftarrow, S_5 \leftarrow NP_4 \wedge NP_6, NP_7 \leftarrow N_8, N_9 \leftarrow, \leftarrow S_10\},$$
$$\{\langle NP_1, NP_4\rangle, \langle N_2, N_3\rangle, \langle NP_6, NP_7\rangle, \langle N_8, N_9\rangle, \langle S_5, S_10\rangle\}\rangle$$

The reader can check for himself that the links obey the constraints formulated in the previous section. We can now start removing links. The following pair of sets defines the graph after deletion of the link $\langle N_2, N_3\rangle$

$$\langle\{NP_1 \leftarrow, S_5 \leftarrow NP_4 \wedge NP_6, NP_7 \leftarrow N_8, N_9 \leftarrow, \leftarrow S_{10}\},$$
$$\{\langle NP_1, NP_4\rangle, \langle NP_6, NP_7\rangle, \langle N_8, N_9\rangle, \langle S_5, S_{10}\rangle\}\rangle$$

The rest of the proof is left to the reader.

Wallen [Wallen, 1990] discusses the concept of Matrix Proof Methods. This work is indirectly related to the connection graphs method. What is interesting is that the matrix proofs can be generalised to several types of logics (with amongst others the intuitionistic propositional logic which is the logical counterpart of the Lambek calculus). We shall not pursue this connection here any further.

6 Conclusion

The results discussed in this paper are part of a larger enterprise: investigating the usefulness of alternative proof procedures for the Lambek calculus. The decomposition of categories was necessary to define a conjunctive normal form for a category. Given such a notion, it is fairly easy to define a GCNF' proof system or a resolution system for the Lambek calculus. The constraints on the table are just a corollary to the whole procedure but are of great potential in the field of parsing with categorial grammars.

References

[van Benthem, 1986] J. van Benthem. *Essays in Logical Semantics.* Reidel, Dordrecht, 1986.

[Gallier, 1987] J. Gallier. *Logic for Computer Science.* John Wiley Sons, New York, 1987.

[Hepple, 1990] M. Hepple. Normal form theorem proving for the lambek calculus, 1990. Ms. Centre for Cognitive Science 7, Cambridge.

[König, 1989] E. König. Parsing as natural deduction. In *Proceedings of the 27th Annual Meeting of the Association of Computational Linguistics*, Vancouver, 1989.

[Kowalski, 1979] Robert Kowalski. *Logic for problem solving.* North-Holland, New York, 1979.

[Lambek, 1958] J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–169, 1958.

[Lambek, 1987] J. Lambek, 1987. a letter published in Michael Moortgat, Richard T. Oehrle, Mary McGee Wood: *Categorial Grammar Newsletter* p. 19.

[Moortgat, 1988] M. Moortgat. *Categorial Investigations. Logical and Linguistic Aspects of the Lambek Calculus.* GRASS. Foris, Dordrecht, 1988. diss. Amsterdam.

[Moortgat, 1990] M. Moortgat. Unambiguous proof representations for the lambek calculus. Ms. OTS, Utrecht, 1990.

[Roorda, 1990] D. Roorda. Proof nets for lambek calculus. Ms. ITLI, Amsterdam, 1990.

[Smullyan, 1968] R. Smullyan. *First-order Logic.* Springer Verlag, Berlin, 1968.

[Wallen, 1990] L. Wallen. *Automated proof search in non-classical logics.* The MIT press, Cambridge (Mass), 1990.

[van der Wouden and Heylen, 1988] T. van der Wouden and D. Heylen. Massive disambiguation of large text corpora with flexible categorial grammar. In *Proceedings of the 12th International Conference on Computational Linguistics.* 1988.