Parsing CCGbank with the Lambek Calculus

Timothy A. D. Fowler
Department of Computer Science
University of Toronto
10 King's College Road, Toronto, ON, M5S 3G4, Canada
tfowler@cs.toronto.edu

May 1st, 2009

Abstract

This paper will analyze CCGbank, a corpus of CCG derivations, for use with the Lambek calculus. We also present a Java implementation of the parsing algorithm for the Lambek calculus presented in Fowler (2009) and the results of experiments using that algorithm to parse the categories in CCGbank. We conclude that the Lambek calculus is computationally tractable for this task and provide insight into a full conversion of CCGbank to a bank of Lambek derivations.

1 Introduction

The Lambek calculus (Lambek, 1958) and Combinatory Categorial Grammar (CCG) (Steedman, 2000) are two related variants of categorial grammar that have received a lot of attention from the computational linguistics community.

The attention that the Lambek calculus has received has been mostly theoretical, with particularly important results including its weak equivalence to context-free grammars (CFGs) (Pentus, 1997) and the NP-completeness of its parsing problem (Savateev, 2008). In response to these results, Tiede (1999) proves that the Lambek calculus and CFGs are not strongly equivalent and Fowler (2009) provides a method for restricting the Lambek calculus to obtain a polynomial time parsing algorithm. CCG research, on the other hand, has been more practically focused. Hockenmaier and Steedman (2007) introduces CCGbank, a corpus of CCG derivations translated from the WSJ section of the Penn treebank. Then, Clark and Curran (2004) describe a wide-coverage statistical parser that using CCGbank for training and that is competitive with other state of the art parsers.

The aim of this paper is to begin to bring these two research streams together. That is, we will investigate the practicality of using the algorithm of Fowler (2009) to parse sentences using the categories of CCGbank. First, we analyze CCGbank for use with the Lambek calculus and present the results of some experiments using a Lambek calculus grammar based on the categories in CCGbank. Then, we provide methods for converting the most problematic derivations in CCGbank to Lambek calculus derivations for use in future research.

One might be tempted to ignore the Lambek calculus in favour of extending standard CCG by introducing rules such as the D-combinator of Hoyt and Baldridge (2008) as they become necessary for linguistic analysis. However, Zielonka (1981) proves that the introduction rules of the Lambek calculus cannot be replaced by any finite number of such rules. Thus, in one step the Lambek calculus gives us a rule system that would require the addition of an infinite number of logically sound combinatory rules.

2 The Lambek Calculus

The set of *categories* $\mathcal{C}$ is built up from a set of *atoms* (e.g. $\{S, NP, N, PP\}$) and the two binary connectives / and \. A *Lambek grammar* G is a 4-tuple $\langle \Sigma, A, R, S \rangle$ where Σ is an *alphabet*, A is a set of atoms, R is a relation between symbols in Σ and categories in $\mathcal{C}$ and S is the set of *sentence categories*.

The order of a category is a measure of its complexity, which measures the depth of the nesting of arguments and is formally defined as follows:

$$\begin{aligned} o(\alpha) &= 0 \text{ for } \alpha \text{ a basic category} \\ o(\alpha/\beta) &= o(\beta\backslash\alpha) = \max(o(\alpha), o(\beta)+1) \end{aligned}$$

For example, the order of $(S/NP)\backslash(S/NP)$ is 2. A *Lambek grammar with order bounded by* k is a Lambek grammar such that R is a relation between symbols in Σ and categories in $\mathcal{C}$ of order $\leq k$.

Parsing with the Lambek calculus is equivalent to logical deduction from the categories of the sentence. The inference rules for the Lambek calculus are shown in figure 1. The $/E$ and $\backslash E$ rules are referred to as *elimination rules*. The $/I_i$ and $\backslash I_i$ are referred to as *introduction rules* and allow the introduction of hypothetical categories (shown in square brackets) with the requirement that they are *discharged* at a point in the proof where they are the rightmost (in the case of $/I_i$) or leftmost (in the case of $\backslash I_i$) category in the proof tree.

$$\frac{X/Y \qquad Y}{X}\,/E \qquad \frac{Y \qquad Y\backslash X}{X}\,\backslash E$$

$$\frac{\begin{array}{c}[Y]_i \\ \vdots \\ X\end{array}}{X/Y}\,/I_i \qquad \frac{\begin{array}{c}[Y]_i \\ \vdots \\ X\end{array}}{Y\backslash X}\,\backslash I_i$$

Figure 1: Inference rules in the Lambek Calculus.

The parsing problem for a string of symbols $s_1 \ldots s_k$ can be characterized as follows: $s_1 \ldots s_k \in \Sigma^*$ is parseable in a Lambek grammar $\langle \Sigma, A, R, S \rangle$ if there exists $c_1, \ldots, c_k \in \mathcal{C}$ such that $c_i \in R(s_i)$ for $1 \leq i \leq k$ and the categories $c_1, \ldots, c_k$ logically derive some category in S via the rules in figure 1. Fowler (2009) provides a chart parsing algorithm that uses abstract term graphs (ATGs) as entries in the chart that is polynomial time if the order of categories is bounded. It is this algorithm that we will be evaluating for practical use with CCGbank.

3 CCG

Combinatory Categorial Grammar (CCG) (Steedman, 2000) is a well-known variant of categorial grammar that incorporates a number of combinatory rules in addition to the *elimination rules* of the Lambek calculus. We will divide the combinatory rules into two groups. Those which are derivable in the Lambek calculus:

- **C**omposition
  1. $X/Y, Y/Z \Rightarrow X/Z$ (Forward)
  2. $Z\backslash Y, Y\backslash X \Rightarrow Z\backslash X$[1] (Backward)
- **T**ype-Raising
  1. $X \Rightarrow T/(X\backslash T)$ (Forward)
  2. $X \Rightarrow (T/X)\backslash T$ (Backward)

And those rules which are not derivable in the Lambek calculus (which we will refer to as non-Lambek rules):

- **C**rossed Composition
  1. $X/Y, Z\backslash Y \Rightarrow Z\backslash X$ (Forward Crossing)
  2. $Y/Z, Y\backslash X \Rightarrow X/Z$ (Backward Crossing)
- **S**ubstitution
  1. $(X/Y)/Z, Y/Z \Rightarrow X/Z$
  2. $Z\backslash(X/Y), Z\backslash Y \Rightarrow Z\backslash X$
  3. $Z\backslash Y, Z\backslash(Y\backslash X) \Rightarrow Z\backslash X$
  4. $Y/Z, (Y\backslash X)/Z \Rightarrow X/Z$

[1] For consistency we will be using the Ajdukiewicz notation, rather than the Steedman notation usually used for CCG.

Any linguistic analysis made within CCG which uses any of the non-Lambek rules will need to be modified to receive an analysis within the Lambek calculus.

4 The Lambek Calculus and CCGbank

Section 4.1 discusses the order of the categories in CCGbank. Section 4.2 discusses an implementation of the algorithm of Fowler (2009) run on the categories of CCGbank. Section 4.3 discusses the necessary major modifications to convert the CCG derivations of CCGbank to Lambek derivations.

4.1 The order of CCGbank

The parsing algorithm described in Fowler (2009) is only polynomial time if the order of categories are bounded by a constant. Furthermore, the algorithm is exponential in that constant. So, the practicality of parsing CCGbank using this algorithm is very dependent on the order of categories in CCGbank. Figure 2 shows the distribution of order across the categories occurring in sections 02 to 21 of CCGbank. The maximum order of any category is 5 and the average is 0.78. These results indicate that the parsing algorithm will be efficient on CCGbank categories, if not competitive with other categorial grammar parsing strategies.

4.2 Parsing CCGbank

We implemented the parsing algorithm of Fowler (2009) in Java and performed our experiments on a machine with 4 Intel Xeon CPUs at 3.0 Ghz and 16Gb of RAM. Our training data was taken from sections 02-21 of CCGbank and our test data was taken from section 00. We have reserved section 23 for future use.

We define Lambek grammars $\langle \Sigma, A, R_t, S \rangle$ where A is the set of atoms appearing in sections 02-21.[2]

[2]We ignore the feature system of CCGbank because the parsing algorithm cannot currently handle the unification of features.

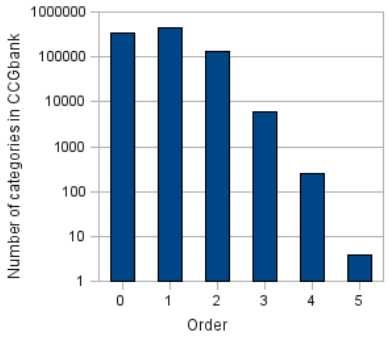

Figure 2: The order of categories in CCGbank

t	Category-Word pairs	Categories per word
0.1	59683	1.350
0.01	69382	1.569
0.001	73265	1.657
0	74667	1.689

Table 1: The effect of the threshold on the lexicon.

Σ is the set of words appearing in both sections 00 and 02-21 except that we will disregard punctuation entirely. R_t will be defined as follows:

- $R_t(s) = \{c | c$ has frequency at least t among the categories for $s\}$ for s in sections 02-21
- $R_t(s) = \{NP, N\}$ for s not in sections 02-21

NP and N were chosen because nouns and proper nouns make up a large portion of unknown words in corpora. The sentence categories are defined as $S = \{S, NP, PP, S/NP, NP\backslash S\}$ which are the root categories of trees in sections 02-21. We will be using the following values for t: 0.1, 0.01 and 0.001. Their effect on the lexicon is shown in table 1.

The results of the experiments are shown in table 2. We can see that only a very small number of sentences receive parses even with a value of $t = 0.001$. The reason for this is the very large number of unary type changing rules from N to NP in CCGbank. Without solving the unary type changing problem in CCGbank, we can increase the number of derivable sentences (and consequently get a more realistic

Grammar Type	Sentences with Parses	Seconds per Sentence	Seconds Total	Mean # of Atoms	Mean # of ATGs
Basic, $t = 0.1$	47/1913	0.365	700	104.4	73.4
Basic, $t = 0.01$	112/1913	0.381	729	241.3	409.7
Basic, $t = 0.001$	247/1913	4.034	7717	611.4	5280.1
$N = NP$, $t = 0.1$	289/1913	0.369	706	104.4	183.8
$N = NP$, $t = 0.01$	678/1913	0.578	1106	241.3	1275.3
$N = NP$, $t = 0.001$	1005/1913	25.044	47908	611.4	14995.5

Table 2: Results of experiments

picture of practical parsing) by converting all occurrences of N to NP (denoted $N = NP$ in figure 2).

Without converting N to NP, parsing is efficient, but this is largely due to a small number of entries in the chart. However, when we equate N and NP, parsing remains efficient for t values of 0.1 and 0.01 and the higher numbers of sentences receiving parses makes these cases a more realistic picture of parsing. The last case ($N = NP$, $t = 0.001$) sees a huge increase in parse time, but also a significant increase in coverage. It is likely that the parse time here would be reduced if we used the supertagging techniques of Clark and Curran (2004). In addition, the low coverage of even our best method is likely due to the categories in CCGbank being designed to participate in non-Lambek rules. This will be discussed in the next section.

4.3 Converting CCGbank

In addition to the usual non-Lambek rules of CCG there are a number of additional rules used in derivations in CCGbank which can be characterized as general phrase structure rules. Due to the nature of parsing algorithms for CCG, these rules do not introduce any additional difficulties for parsing CCGbank using CCG but they cannot be easily accommodated into a logical approach such as the Lambek calculus. In this section, we analyze the non-Lambek rules of CCGbank and discuss methods for converting these analyses to ones using Lambek's rules.

Table 3 categorizes the binary rules in CCGbank according to their type. The first 6 rule types are the rules from CCG and the last 3 are general phrase structure rules. The group of Crossing Composition rules are of concern because they are large in number and because they are a fundamental part of CCG. The vast majority of such rules attach lexical adjuncts to items that they modify. The directions of the slashes in these lexical items can be modified so that this is no longer crossing composition. In some cases, this will increase lexical ambiguity, but this lexical ambiguity appears necessary unless we adopt some kind of crossing rules into the Lambek calculus. The "Punctuation and Conjunction" rules are rules for handling various kinds of punctuation and conjunctions such as "and" and "or". One way that these rules can be handled is to assign certain categories to punctuation marks (Hockenmaier, 2003, pg. 34) and assigning coordination categories such as $(NP\backslash NP)/NP$ to the conjunctions coordinating NPs. The small number of rules under the "Merger" category are rules that take two identical atoms and produce one instance of the same atom. These can be altered on a case by case basis since there are so few and most can be modified so that the rule becomes an application rule. Similarly, for the "Other" rules.

Table 4 groups the unary rules of CCGbank into categories. The Type-Raising rules are Lambek rules, but the remainder must be altered to be considered in a Lambek framework. Of these unary rules, the vast majority are of the type $N \Rightarrow NP$ meaning that within the derivation we need to convert some category's instance of an N atom into an NP atom. These can easily be changed in the lexicon, increasing lexical ambiguity but allowing for a more

Rule Type	# in CCGbank
Forward Application	674513
Backward Application	227663
Forward Composition	7947
Backward Composition	1844
Crossing Composition	14468
Substitution	4
Punctuation and Conjunction	172987
Mergers	32
Other	6
Total	1099464

Table 3: Counts of Binary Rules

Rule Type	# in CCGbank
Type-Raising	4016
$N \Rightarrow NP$	142525
$NP\backslash S \Rightarrow NP\backslash NP$	8986
$NP\backslash S \Rightarrow (NP\backslash S)\backslash(NP\backslash S)$	3421
Other	4690
Total	163638

Table 4: Counts of Unary Rules

classical categorial analysis. In addition, there are rules for handling relative clauses of the form $S/NP \Rightarrow NP\backslash NP$. These rules can be handled in a similar way, but the effect on the size of the lexicon is an open question.

5 Conclusion

Our experiments indicate that the NP-completeness of parsing with the Lambek calculus is not the barrier that it seems to be, given the fact that our implementation is basic and can be improved in a number of ways. The low coverage of the parser is problematic, but is likely due to the fact that we have not yet fully converted CCGbank for use with the Lambek calculus. To address this, we have outlined methods for doing such a conversion.

However, the benefits of using the Lambek calculus as a grammar over a combinatory categorial grammar cannot be fully realized until the derivations in the corpus take advantage of the logical basis of Lambek's introduction rules. This can be accomplished by revisiting the methods used to convert the Penn treebank to CCG by Hockenmaier and Steedman (2007).

References

S. Clark and J. R. Curran. Parsing the WSJ using CCG and log-linear models. *Proceedings of ACL '04*, pages 104–111, 2004.

T.A.D. Fowler. A Polynomial Time Algorithm for Parsing with the Bounded Order Lambek Calculus. In *Proceedings of MOL '09*, 2009.

J. Hockenmaier. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh., 2003.

J. Hockenmaier and M. Steedman. CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396, 2007.

F. Hoyt and J. Baldridge. A Logical Basis for the D Combinator and Normal Form in CCG. *Proceedings of ACL '08*, 2008.

J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958.

M. Pentus. Product-Free Lambek Calculus and Context-Free Grammars. *The Journal of Symbolic Logic*, 62(2):648–660, 1997.

Y. Savateev. Product-free Lambek Calculus is NP-complete. *CUNY CS Tech Report*, 2008.

M. Steedman. *The Syntactic Process*. Bradford, 2000.

H.J. Tiede. *Deductive Systems and Grammars: Proofs as Grammatical Structures*. PhD thesis, Indiana University, 1999.

W. Zielonka. Axiomatizability of Ajdukiewicz-Lambek calculus by means of cancellation schemes. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, 27, 1981.