A Polynomial Time Algorithm for Parsing with the Bounded Order Lambek Calculus

Timothy A. D. Fowler
Department of Computer Science
University of Toronto
10 King's College Rd., Toronto, ON, M5S 3G4, Canada

tfowler@cs.toronto.edu

Abstract

Fowler (2008) introduced the bounded-order Lambek Calculus and provided a polynomial time algorithm for its sequent derivability. However, this result is limited because it requires exponential time in the presence of lexical ambiguity. That is, Fowler (2008) did not provide a polynomial time *parsing* algorithm. The purpose of this paper will be to provide such an algorithm. We will prove an asymptotic bound of $O(n^4)$ for parsing and improve the bound for sequent derivability from $O(n^5)$ to $O(n^3)$.

1 Introduction

The Lambek calculus (L) (Lambek, 1958) is a categorial grammar formalism having a number of attractive properties for modelling natural language. In particular, it is strongly lexicalized which allows the parsing problem to be a problem over a local set of categories, rather than a global set of rules and there is a categorial semantics accompanying every syntactic parse.

Despite these attractive properties, Pentus (1997) proved that L is weakly equivalent to context-free grammars (CFGs) which are widely agreed to be insufficient for modelling all natural language phenomena. In addition, Savateev (2009) proved that the sequent derivability problem for L is NP-complete. The weak equivalence to CFGs has been addressed by a number of authors. Kruijff and Baldridge (2000) introduced a mildly context-sensitive extension of L, increasing the weak equivalence of the formalism and Tiede (1999) proved that L is more powerful than CFGs in terms of strong generative capacity. Moot (2008) proved that restrictions of the multi-modal non-associative Lambek calculus is mildly context-sensitive, which raises interesting questions as to the importance of associativity to parsing.

We will address the issue of parsing complexity by extending the results of Fowler (2008) and providing a *parsing* algorithm for the Lambek calculus which runs in $O(n^4)$ time for an input of size n when the order of categories is bounded by a constant. The key to this result is in restricting Fowler's representation of partial proofs to be local to certain categories rather than global. Our results are

for the product-free fragment of the Lambek calculus (L) and the variant that allows empty premises (L^*), for simplicity and because the product connective has limited linguistic application. This work can be also seen as a generalization of Aarts (1994) which proved that if the order of categories is less than two then polynomial time sequent derivability is possible.

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*. A *sequent* is a sequence of categories (called the *antecedents*) together with the $\vdash$ symbol and one more category (called the *succedent*).

The parsing problem for a string of symbols $s_1 \dots s_l$ can be characterized in terms of the sequent derivability problem as follows: $s_1 \dots s_l \in \Sigma^*$ is parseable in a Lambek grammar $\langle \Sigma, A, R, S \rangle$ if there exists $c_1, \dots, c_k \in \mathcal{C}$ and $s \in S$ such that $c_i \in R(s_i)$ for $1 \leq i \leq k$ and the sequent $c_1 \dots c_k \vdash s$ is derivable in the Lambek calculus. The sequent derivability problem can be characterized by conditions on *term graphs* (Fowler, 2009). A term graph for a sequent is formed in a two step process.

The first step is deterministic and begins with associating a polarity with each category in the sequent: Negative for the antecedents and positive for the succedent. Next, we consider the polarized categories as vertices and decompose the slashes according to the following vertex rewriting rules:

$$\begin{array}{lcl} (\alpha/\beta)^- & \Rightarrow & \alpha^- \rightarrow \beta^+ \\ (\beta\backslash\alpha)^- & \Rightarrow & \beta^+ \leftarrow \alpha^- \\ (\alpha/\beta)^+ & \Rightarrow & \beta^- \dashleftarrow \alpha^+ \\ (\beta\backslash\alpha)^+ & \Rightarrow & \alpha^+ \dashrightarrow \beta^- \end{array}$$

The neighbourhood of the polarized category on the left of each rule is assigned to α. The dashed edges are called *Lambek edges* and the non-dashed edges are called *regular edges*. This process translates categories into trees. Then, additional Lambek edges are introduced from the root of the succedent tree to the roots of the antecedent trees which are referred to as *rooted* Lambek edges. The result of the graph rewriting is an ordered sequence of polarized atoms.

The second step is non-deterministic and assigns a complete *matching* of the polarized atoms. The matching must be planar above the atoms and match occurrences of atoms only with other occurrences of the same atom but of opposite polarity. These pairings of atom occurrences are called *matches*. The edges in the matching are regular edges and are directed from the positive atom to the negative atom. An example of a term graph can be seen in figure 1.

We will prefix the term *regular* (resp. *Lambek*) to the usual definitions of graph theory when we mean to refer to the sub-graph of a term graph obtained by restricting the edge set to only the regular (resp. Lambek) edges.

A term graph is L^**-integral* if it satisfies the following conditions:

1. G is regular acyclic.

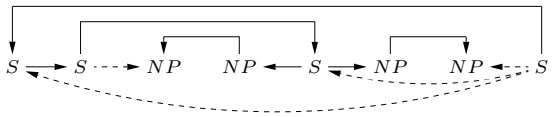

Figure 1: An integral term graph for the sequent $S/(NP\backslash S), (NP\backslash S)/NP, NP \vdash S$.

2. For every Lambek edge $\langle s, t\rangle$ in G there is a regular path from s to t.

A term graph is *integral* if it is L^*-integral and it satisfies the following:

3. For every Lambek edge $\langle s, t\rangle$ in G, there is a regular path from s to a vertex x in G such that if x has a non-rooted Lambek in-edge $\langle s', x\rangle$ then there is no regular path from s to s'.

Theorem 2.1. *A sequent is derivable in L iff it has an integral term graph. A sequent is derivable in L^* iff it has an L^*-integral term graph.*

Proof. See Fowler (2009). □

3 Parsing with L^*

Fowler (2008) presented a chart parsing algorithm that uses an abstraction over term graphs[1] as a representation and considers the exponential number of matchings by building them incrementally from smaller matchings. The indices of the chart are the atoms of the sequent. Applying this algorithm to the parsing problem is exponential time because near the bottom of the chart, there would need to be one entry in the chart for every possible sequence of categories. We will generalize this algorithm to the parsing problem by restricting the domain of the abstractions to be the categories of the atoms at its endpoints.

3.1 Abstract Term Graphs

In this section, we introduce versions of the L^*-integrity conditions that can be enforced incrementally during chart parsing and using only the information contained in the abstractions.

A *partial matching* is a contiguous sub-matching of a complete matching. The notion of *partial term graph* is defined in the obvious way. In a partial term graph, a vertex is *open* if it does not have a match. A partial term graph is *L^*-incrementally integral* if it satisfies the following:

1. G is regular acyclic

2. For every Lambek edge $\langle s, t\rangle$ either there is a regular path from s to t or there is an open positive vertex u such that there is a regular path from s to u and there is an open negative vertex v such that there is a regular path from t to v.

[1]There are some minor differences between the presentations. For discussion, see Fowler (2009).

The intuition is that a partial term graph G is L^*-incrementally integral exactly when extending its matching to obtain an L^*-integral term graph has not already been ruled out. It should be easy to see that L^*-incrementally integral term graphs are L^*-integral. We will only insert representations into the chart for partial term graphs that are L^*-incrementally integral.

An *abstract term graph* $\mathcal{A}(G)$ is obtained from an L^*-incrementally integral term graph G by performing the following operations:

1. Deleting Lambek edges $\langle s, t\rangle$ s.t. there is a regular path from s to t.
2. Replacing Lambek edges $\langle s, t\rangle$ where t is not open by $\langle s, r\rangle$ where r is the open negative vertex from which there is a regular path to t.
3. Replacing Lambek edges $\langle s, t\rangle$ where s is not open by edges $\langle r, t\rangle$ for $r \in R$ where R is the set of open positive vertices with regular paths from s.
4. Contracting regular paths between open vertices to a single edge and deleting the non-open vertices on the path

For the term graph with the empty matching G_ϵ, $\mathcal{A}(G_\epsilon) = G_\epsilon$. Planar matchings are built out of smaller partial planar matchings in one of two ways: *Bracketing* and *Adjoining*. Bracketing a partial matching extends it by introducing a match between the two atoms on either side of the matching. Adjoining two partial matchings concatenates the two partial matchings to obtain a single larger matching.

Fowler (2008) details two sub-algorithms for bracketing and adjoining ATGs. Given an ATG $\mathcal{A}(G)$, Fowler's bracketing algorithm determines whether extending G by bracketing (for any term graph G with ATG $\mathcal{A}(G)$) results in an incrementally integral term graph H and returns the ATG $\mathcal{A}(H)$ if it does. Given two ATGs $\mathcal{A}(G_1)$ and $\mathcal{A}(G_2)$, the adjoining algorithm does the same for adjoining.

3.2 The parsing algorithm

Polynomial time parsing depends on the following key insight: the only information in ATGs needed to run the bracketing and adjoining algorithms is the edges between atoms originating from *categories at the endpoints* of the ATG.

Let $G = \langle \Sigma, A, R, S\rangle$ be a Lambek grammar and let $s_1 \dots s_l \in \Sigma^*$ be the input string. Because an atom or category can occur more than once in the input, we need the notions of *category occurrence* and *atom occurrence* to be specific instances of a category and an atom, respectively. Given a category occurrence c, the *partial term graph* for c, denoted $T(c)$, is the graph obtained from the deterministic step of term graph formation without the rooted edges. The *atom sequence* for c, denoted $A(c)$, is the sequence of atom occurrences of $T(c)$. Given an atom occurrence a, $C(a)$ is the category occurrence from which it was obtained and given a category occurrence c, $S(c)$ is the symbol s_i from which it was obtained.

Our new definition of ATGs is as follows: For a partial term graph G, whose matching has endpoints a_s and a_e, its ATG $\mathcal{A}(G)$ is obtained as before but with the following final step:

5. Deleting all vertices v such that $C(v) \neq C(a_s)$ and $C(v) \neq C(a_e)$

The bracketing and adjoining algorithms of Fowler (2008) require knowledge both about the ATGs that they are bracketing and adjoining but also the state of the term graphs before any matchings have been made, which is referred to as the *base ATG*. In our case, the base ATG is simply the union of the term graphs of all categories of all symbols. In addition, Fowler (2008) includes labels on the Lambek edges of ATGs that are the same as the target of the Lambek edge in the base ATG. Section 4 details a simpler and more elegant method of representing Lambek edges in ATGs. Finally, since the partial term graphs for categories do not include the rooted Lambek edges, they will need to be inserted whenever an ATG, G_2, with a right endpoint in a succedent category is combined with an ATG, G_1, with a left endpoint in a different category than the left endpoint of G_2. Then, Lambek edges are inserted from the positive vertices with in-degree 0 in G_1 to the negative vertex with in-degree 0 in G_2.

Our chart will be indexed by atom occurrences belonging to categories of symbols in the input string and categories in S. Entries consist of sets of ATGs and each entry $\langle a_s, a_e \rangle$ will be specified by its leftmost matched atom a_s and its rightmost matched atom a_e. A chart indexed in this way can be implemented by arbitrarily ordering the categories for a symbol and keeping track of the locations of word and category boundaries in the sequence of atoms. Sections 3.2.1 and 3.2.2 will outline the insertion of ATGs into the chart.

3.2.1 Inserting ATGs for minimal matchings

A *minimal matching* is a matching consisting of exactly one match. Minimal matchings are inserted in two distinct steps. First, we insert minimal matchings over atom occurrences a_s and a_e where $C(a_s) = C(a_e) = C$ and a_s and a_e are adjacent in C. The ATG input to the bracketing algorithm is $T(C)$.

Second, we insert minimal matchings over atoms a_s and a_e where $C(a_s) \neq C(a_e)$, a_s is the rightmost atom in $C(a_s)$ and a_e is the leftmost atom in $C(a_e)$. The ATG input to the bracketing algorithm is $T(C(a_s)) \cup T(C(a_e))$.

3.2.2 Inserting ATGs for non-minimal matchings

Once we have inserted the ATGs for minimal matchings, we must *process* each ATG in the chart. That is, we consider all possible ways of bracketing and adjoining each ATG in the chart, which may require insertion of more ATGs that, in turn, need to be processed.

An entry $\langle a_s, a_e \rangle$ *left subsumes* an entry $\langle a_t, a_f \rangle$ if one of the following are satisfied:

1. a_t is equal to a_s
2. $C(a_t) = C(a_s) = C$ and a_t appears to the right of a_s in $A(C)$
3. $C(a_t) \neq C(a_s)$ and $S(C(a_t)$ appears to the right of $S(C(a_s))$ in the input string

The notion of *right subsumes* is defined similarly for a_f and a_e. An entry E *subsumes* an entry F if it left subsumes and right subsumes F. The intuition is that E subsumes F iff the endpoints of F appear between the endpoints of E.

The *size* of an entry $\langle a_s, a_e \rangle$ is the distance between a_s and a_e, if $C(a_s) = C(a_e)$ and the distance between $S(C(a_s))$ and $S(C(a_e))$, if $C(a_s) \neq C(a_e)$.

However, any category with endpoints in the same category is smaller than any category for which they are not.

We must take care not to process an entry until all ATGs for that entry have been inserted. All ATGs for an entry E have been inserted only after all entries that E subsumes have been processed. To ensure this, we process entries as follows. First, we process entries whose endpoints are in the same category. We process entries from smallest to largest and among entries of equal size from left to right. Second, we process entries whose endpoints are in different categories in the same order.

To process an entry $E = \langle a_s, a_e \rangle$, we must consider all possible bracketings and adjoinings. To do this, we first calculate the set of atoms $L(a_s)$ and $R(a_e)$, which will correspond to atoms which could occur to the left of a_s in a sequent and to the right of a_e, respectively. If a_s is the leftmost atom in its category, then $L(a_s)$ is the set of atoms such that they are the rightmost atom in a category occurrence for the symbol occurring to the left of $S(C(a_s))$ in the input string. If $S(C(a_s))$ is the leftmost symbol in the input string then $L(a_s) = \emptyset$. If a_s is not the leftmost atom in its category, then $L(a_s)$ is the singleton consisting of the atom to the left of a_s in $A(C(a_s))$. $R(a_e)$ is computed analogously. Then, let G be an ATG in E and let $a_l \in L(a_s)$ and $a_r \in R(a_e)$. First, we run the bracketing algorithm with input G, a_l and a_r. Then, for each ATG H in an entry F where $F = \langle a, a_l \rangle$ or $F = \langle a_r, a \rangle$ for some atom a and such that F is smaller than E, we run the adjoining algorithm with input G and H. Lastly, for each ATG H in an entry the same size as E with endpoint a_l, we run the adjoining algorithm with input G and H. When running the bracketing and adjoining algorithms we insert the output into the chart at the appropriate place when they are successful.

After processing every entry, we output "YES" if there are any ATGs in any entry $\langle a_s, a_e \rangle$ such that a_s is the leftmost atom in a category in s_1 and a_e is the rightmost atom in a category in S. Otherwise, we output "NO".

3.2.3 Correctness

It is not hard to prove that the chart iteration process in the preceding sections considers every possible matching over every possible sequent for the sentence. Then, the bracketing and adjoining algorithms of Fowler (2008) ensure any ATG inserted is the ATG of an L^*-incrementally integral partial term graph. Finally, any ATG in an entry whose left endpoint is the leftmost atom of a category for s_1 and whose right endpoint is the rightmost atom of a category for S is the ATG of an L^*-incrementally integral term graph over some sequence of categories which must be an L^*-integral term graph.

4 Running time

Fowler (2008) introduced the *bounded order Lambek calculus* where the categories in the input have order bounded by k. In the context of term graphs, bounding order by k is equivalent to bounding the length of paths in the base ATG. The key insight for achieving polynomial time is that bounding the lengths of paths in partial term graphs bounds the lengths of paths in ATGs in the chart, which in turn, bounds the variation.

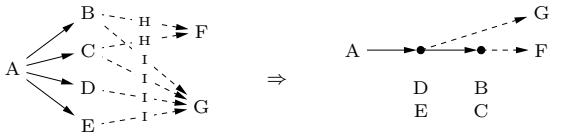

Figure 2: Comparing ATG representations

To improve the bound and simplify the representation, we introduce a better representation of ATGs. In the original definition of ATGs, the out-neighbourhood of negative vertices is not structured and the applicability of the integrity conditions during bracketing and adjoining is informed by the labels on Lambek edges. However, from ATG $\mathcal{A}(G)$ on the left of figure 2, we can deduce that for Lambek edges $\langle X, H\rangle$ and $\langle Y, I\rangle$ in G, there must be a regular path from Y to X. If we maintain this structure explicitly, then we can remove the need for labels on Lambek edges, since the labels are only used to deduce this structure. Furthermore, we can observe that any two positive vertices in an ATG who share an in-neighbour in the base ATG must have identical neighbourhoods in the ATG. Rather than representing each such positive vertex distinctly, we can use a placeholder that points to sets of positive vertices sharing in-neighbours in the base ATG. These two modifications are shown in figure 2 where the sets are listed below the placeholders.

Finally, observe that any partition of the atoms of a category results in a path in the base ATG. Given an entry $\langle a_s, a_e\rangle$, the partition of the atoms to the left of a_s and to the right of a_e results in two such paths. Then, any two ATGs in $\langle a_s, a_e\rangle$, can differ only by the neighbourhoods they assign to vertices incident to edges on one of the two paths.

Bounded order gives us bounded paths and our new representation bounds the branching factor on these paths which means that the sub-graph of an ATG which is manipulated during bracketing and adjoining is of a constant size. Therefore, adjoining and bracketing are constant time operations. Also, the number of ATGs for an entry must also be constant, since the variation within them is bounded by a constant.

Let n be the number of atoms in categories of symbols in the input. Then, the chart is of size $O(n^2)$ and while processing an entry, we bracket $O(n)$ times and adjoin with the ATGs in up to $O(n^2)$ entries. Thus, the running time of our algorithm is $O(n^4)$. Without lexical ambiguity, we only need to adjoin with ATGs in $O(n)$ entries yielding a time bound of $O(n^3)$ for sequent derivability.

5 Parsing with L

To ensure correctness of the algorithm with respect to L, we need a notion of incremental integrity for partial term graphs. A partial term graph G is incrementally integral if it is L^*-incrementally integral and satisfies the following:

3. For every Lambek edge $\langle s, t\rangle$ in G either there is a regular path from s to a negative vertex x such that if x has a non-rooted Lambek in-edge $\langle s', x\rangle$ then there is no regular path from s to s' or there is an open positive vertex u such that there is a regular path from s to u and there is a negative

vertex v and an open negative vertex w such that there is a regular path from w to v and if v has a non-rooted Lambek in-edge $\langle s'', v\rangle$ then there is no regular path from w to s''.

To enforce this condition during chart parsing, we will need to associate an *empty premise bit* to positive vertices in ATGs. The bit associated to v_p will represent whether there is a regular path from the source of a Lambek edge that has not yet met condition 3 to v_p in the term graph.

The empty premise bits for an ATG can be calculated analogously to the calculation of Lambek edges during adjoining and bracketing, since they represent the same notions of paths in underlying term graphs.

6 Conclusion

We have provided a parsing algorithm for both the Lambek calculus and the Lambek calculus allowing empty premises when the order of categories is bounded by a constant that runs in time $O(n^4)$. Also, we reduced the asymptotic bound for sequent derivability of Fowler (2008) from $O(n^5)$ to $O(n^3)$. To the best of our knowledge, no linguistic analysis has called for categories of high order which means that these results allow for practical parsing with the Lambek calculus.

References

Erik Aarts. Proving Theorems of the Second Order Lambek Calculus in Polynomial Time. *Studia Logica*, 53:373–387, 1994.

Timothy A.D. Fowler. Efficient Parsing with the Product-Free Lambek Calculus. In *Proceedings of COLING '08*, 2008.

Timothy A.D. Fowler. Term Graphs and the NP-completeness of the Product-Free Lambek Calculus. In *Proceedings of the 14th Conference on Formal Grammar*, 2009.

G.J.M. Kruijff and J.M. Baldridge. Relating categorial type logics and CCG through simulation. *Unpublished manuscript, University of Edinburgh*, 2000.

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

R. Moot. Lambek Grammars, Tree Adjoining Grammars and Hyperedge Replacement Grammars. In *Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms*, 2008.

Mati 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. In *Proceedings of the 2009 International Symposium on Logical Foundations of Computer Science*, pages 380–394. Springer-Verlag Berlin, Heidelberg, 2009.

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