A Generative Process for Lambek Categorial Proof Nets

Jinman Zhao and Gerald Penn
Department of Computer Science
University of Toronto
Toronto, Canada
{jzhao,gpenn}@cs.toronto.edu

Abstract

In this work, we present a stochastic, generative model for Lambek categorial proof sequents (Lambek, 1958). When a set of primitive categories is provided, this model, called PLC is able to generate all sequents of categories that are derivable in the Lambek Calculus with it. We also introduce a simple method to numerically estimate the parameters of the model from an annotated corpus. Then we compare our model with probabilistic context-free grammars (PCFGs). We show that there are several trade-offs with respect to using PLC in place of PCFG. Overall, PLC provides a layer of generalization in exposing numerical parameters of the formalism that is not directly accessible to PCFGs.

1 Introduction

Stochastic variants of different grammars have been proposed over the last several decades, and stochastic methods are very important for natural language processing. For example, stochastic context-free grammars (Huang and Fu, 1971), also known as probabilistic context-free grammar(PCFG), assign a probability to each production rule, normalized over their left-hand sides. Maximum Likelihood Estimation (MLE) and the Inside-Outside algorithm (Lari and Young, 1991) can be used to estimate these rule probabilities, given a training set. Other stochastic models have been proposed for Combinatory Categorial Grammar (Osborne and Briscoe, 1997). Bonfante and de Groote (2004) proposed a stochastic model for Lambek Categorial Grammar. In their work, probabilities are assigned to each leaf of a proof net, with the interpretation that a leaf will appear as the left conclusion of an axiom link with this probability. Probabilities are not attached to derivation rules and their model is fully lexicalized. It is not, however, generative, in the sense that it cannot deterministically produce proof nets from no input. In this respect, it more resembles a supertagger or an automaton-based probabilistic model, such as those that have been proposed for TAG (Bangalore and Joshi, 2010) or dependency grammar (Kübler et al., 2009). These are very useful for parsing. On the other hand, it would still be very instructive to have a generative process for Lambek proof nets more akin to a PCFG.

Pentus proved that both Lambek Grammars and product-free Lambek Grammars are context-free (Pentus, 1993, 1997). Using those constructive proofs, it is possible to convert a PCFG to a generative model of Lambek sequent derivability. The conversion takes exponential time as a function of the original PCFG's size, however, and what the numerical parameters of the PCFG correspond to in terms of proof nets provides little additional illumination. What we propose here is a "native" generative process defined directly for Lambek proof nets.

While the incremental enforcement of certain semantic criteria as necessary side conditions to proof-net construction (Roorda, 1991) is very difficult when a candidate sequent is known at the outset (leading, in one view, to the NP-completeness of the sequent derivability problem), in the generative orientation, it turns out not to be so difficult, as we show below. We also provide a simple parameter estimation method, and compare the results of training a PLC model through MLE to those of training a PCFG on an analogous annotated corpus.

2 Preliminaries

2.1 Lambek Calculus

The Lambek calculus was introduced by Lambek (1958). Given a set of primitive types, $Prim = \{p_1, p_2, p_3, \ ...\}$, and, in this work, only the two connectives $\backslash, /$, we have the following rules:

$$\frac{\Gamma X \to Y}{\Gamma \to Y/X}\ (/R)$$

$$\frac{X\Gamma \to Y}{\Gamma \to X\backslash Y}\ (\backslash R)$$

$$\frac{\Gamma \to X \quad \Delta Y\Theta \to Z}{\Delta Y/X\Gamma \to Z}\ (/L)$$

$$\frac{\Gamma \to X \quad \Delta Y\Theta \to Z}{\Delta\Gamma X\backslash Y\Theta \to Z}\ (\backslash L)$$

$$\frac{\Gamma \to X \quad \Delta X\Theta \to Y}{\Delta\Gamma\Theta \to Y}\ (CUT)$$

along with the axioms, $p_i \to p_i$, for primitive types only. Note that we use the formulation of Pentus (1993). His restriction of these axioms to primitives will be important to us below. The primitive types, together with their closure under the available connectives, will be called *categories*.

We take Lambek Categorial Grammar(LCG) to be a 3-tuple $G = (\Sigma, D, f)$, where Σ is a finite alphabet, D is a distinguished category, and f is a mapping of members of the alphabet $t \in \Sigma$ to a single category, where the possible categories have been derived from the set *Prim* by induction over the connectives. Note that this is a departure from Pentus (1993), in that there is no lexical ambiguity here, as our input will consist of candidate proof sequents, in which a single category is already known for each word. In general, $f(t)$ is a finite set of categories. Lambek (1958) did not define lexical mappings.

2.2 Proof Nets

Roorda (1991) adapted the proof nets of linear logic to the Lambek calculus. The treatment here is taken from Penn (2004). A (Lambek) proof net consists of a lexically unfolded sequence of terminal formulae, a spanning linkage of the resulting sequence of axiomatic formulae and a variable substitution.

As a running example of how to parse with a proof net, let us consider the sequent:

$$S/(NP\backslash S) \quad (NP\backslash S)/NP \quad NP \models S$$

2.2.1 Labelled Terms

Given a sequent, the first step is to add polarities to each category. By convention, all LHS categories receive negative polarity and the RHS category receives positive polarity. We also label each formula with a variable. The result on our example would be:

$$S/(NP\backslash S)^{-}{:}a \quad (NP\backslash S)/NP^{-}{:}b \quad NP^{-}{:}c \quad S^{+}{:}d$$

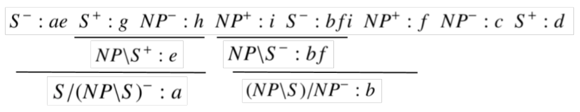
2.2.2 Lexical Unfolding

We then apply the following substitution rules to each labelled category from the previous step until no more rules can be applied:

$$(A\backslash B)^{-}{:}t \to A^{+}{:}u \;\; B^{-}{:}tu$$
$$(A\backslash B)^{+}{:}v \to B^{+}{:}v' \;\; A^{-}{:}u[v := \lambda u.v']$$
$$(A/B)^{-}{:}t \to A^{-}{:}tu \;\; B^{+}{:}u$$
$$(A/B)^{+}{:}v \to B^{-}{:}u \;\; A^{+}{:}v'[v := \lambda u.v']$$

Here, we are expanding every labelled category into a string of labelled categories until only primitive labelled types remain. Note that when positive-polarity categories unfold, they add new variables, u and v'. We will refer to v as a *lambda node* or *lambda variable* when this happens, in honour of the correspondence between these labels and lambda-terms in the corresponding labelled deduction system (Roorda, 1991). u and v' will likewise be referred to as the *daughters* of v.

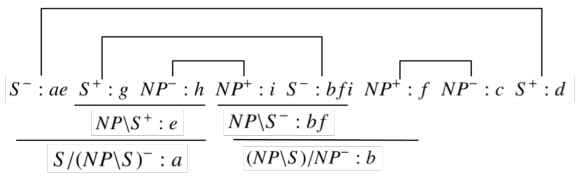
The above example produces:


$S^- : ae$ $S^+ : g$ $NP^- : h$ $NP^+ : i$ $S^- : bfi$ $NP^+ : f$ $NP^- : c$ $S^+ : d$
$NP\backslash S^+ : e$
$NP\backslash S^- : bf$
$S/(NP\backslash S)^- : a$
$(NP\backslash S)/NP^- : b$

Note that negative-polarity categories are generally labelled with strings of variables, not merely one variable.

2.2.3 Axiomatic Linkage

Next, we must link matching primitives of opposite polarities together in a half-planar graph. Each polarized primitive must be linked exactly once. One possible linkage for the above example is:


$S^- : ae$ $S^+ : g$ $NP^- : h$ $NP^+ : i$ $S^- : bfi$ $NP^+ : f$ $NP^- : c$ $S^+ : d$
$NP\backslash S^+ : e$
$NP\backslash S^- : bf$
$S/(NP\backslash S)^- : a$
$(NP\backslash S)/NP^- : b$

The edges of this half-planar graph will be known as *axiom links*.

2.3 LC-graph construction

The first two steps in this process are guaranteed to succeed exactly once. The third step is not — a half-planar linkage may not exist or more than one may exist. Furthermore, even if one does exist, the fourth step below, which can be undertaken in parallel, may fail, requiring the search for more half-planar linkages to continue.

Given a sequence of axiomatic formulae and a partial linkage, every link induces the addition of one or more edges to an LC-graph, in which every node corresponds to a single variable in the category labels.

Let the candidate sequent's LC-graph be a directed graph $G = \langle V, E \rangle$, such that V is the set of all variables that appear in its category labels and E is the smallest set such that:

- for every $v \in V$, if v is a lambda-variable, then for both daughter variables of v, u and v', $(v, u) \in E$, and $(v, v') \in E$, and
- for every axiom link matching $p^+ : u$ and $p^- : t$ and for every v in the string t, $(u, v) \in E$.

The LG-graph for above linkage and sequence of axiomatic formulae is:

$$\begin{tikzcd}
 & a^- & & & \\
d^+ \arrow[ur] \arrow[r] \arrow[dr] & e^+ \arrow[r] & b^- & & \\
 & f^+ \arrow[r] \arrow[drrr, bend right] & g^+ \arrow[r] \arrow[dr] & c^- & \\
 & & & i^+ \arrow[r] & h^-
\end{tikzcd}$$

2.3.1 Integrity Criteria

Penn (2004) proved that an LCG sequent is derivable iff the following three criteria are true of the LC graph, G, of some axiomatic linkage:

- **I(1)** there is a unique node in G with in-degree 0, from which all other nodes are path-accessible.
- **I(2)** G is acyclic.
- **I(3)** for every lambda-node $v \in V$, there is a path from its plus-daughter, v', to its minus-daughter, u.
- **I(CT)** for every lambda-node $v \in V$, there is a path in G, $v \rightsquigarrow x$, such that x labels a negative-polarity category, x has out-degree 0 and there is no lambda-node $v' \in V$ such that $v \rightsquigarrow v' \to x$.

The second criterion can be enforced incrementally. In general, it appears that the others may result in a violation that would cause backtracking and the selection of other axiomatic links in the third step. The value of annotating proof nets with probabilities inheres in its ability to direct us towards axiom links that are likely to lead to the successful construction of a proof net.

3 Generative Process

A generative process is not necessarily just for parsing. It can begin with no input whatsoever, and generate a derivable sequent, along with a numerical score that could correspond to the probability of a corresponding string of words. It can also be made to backtrack after every output sequent, thereby enumerating an infinite sequence of derivable sequents.

Note that in our formulation of the generative process here, however, there is no guarantee that a sequent in fact corresponds to a string of words. To achieve this, we must condition our process to stop only at categories that are attested in the lexicon. What we present here is a generative process for derivable sequents.

To generate any well-formed sequent, we may begin only with $p \to p$, for a primitive type p. Note that p is not required to be D, the distinguished category of an intended grammar, and because there is no requirement that D be primitive, some choices of D will not be allowed here. Nevertheless, because well-formed proof nets are closed under cyclic permutations of their polarized primitive categories, we can assume without loss of generality that the right-hand side of the generated sequent is a primitive category.

Our generative process proceeds in two phases. Both expand the proof net as well as its corresponding LC graph. The first phase adds non-lambda nodes to the LC graph, and the second phase adds lambda nodes to the LC graph. Corollary 4.14 will prove that any proof net can be generated in this fashion: all of the non-lambda nodes first, followed by all of the lambda nodes.

Adding non-lambda nodes affects the number of axiomatic formulae; adding lambda nodes does not. Unlike the method outlined in the previous section for derivability when an input candidate sequent is given, the correctness conditions of the LC-graph will be satisfied throughout the derivation. They will therefore be satisfied in the final proof net.

We will need the following definition for growing larger categories from smaller categories:

Definition 3.1 (Term tree)**.** *Every labelled category in a sequent corresponds to a binary tree of labelled categories. Each subtree in the term tree is rooted in a labelled category $X^{+/-} : v$, where X is a category and v is a sequence of variables. Every interior subtree has a positive child and a negative child, and is licensed by one of the rules shown in Section 2.2.2.*

Figure 1 shows the term tree of $(A/B)/C^- : a$ as an example. Unlike Section 2.2.2, we will think of term trees as corresponding not to top-down unfoldings of complex categories, but to bottom-up compositions of more complex categories from simpler categories.

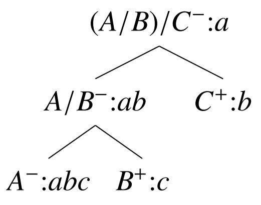
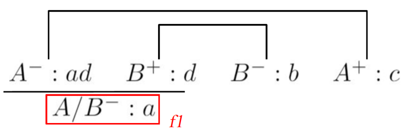


Figure 1: An example term tree.

3.1 Adding Non-lambda Nodes

This phase of the process consists of the iteration of the following steps in sequence zero or more times, until we elect to stop. Each iteration will add one more link and one more matching pair of axiomatic primitives (of opposite sign).

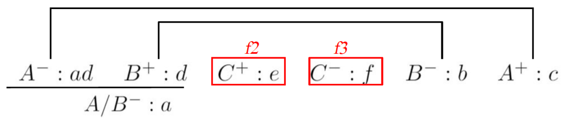
Suppose, for example, that we have already derived the following proof net:



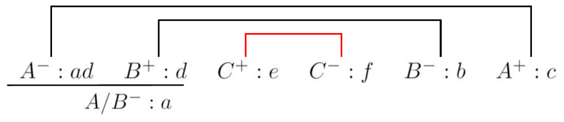
We first pick any negative term (not a subterm) from the existing sequent, e.g., $f_1 = A^- : a$, as the redex. Then we add two new, adjacent polarized primitives, connected by a new link of distance 1 to the immediate left or immediate right of that redex. Both new primitives will have a different, new variable as their labels. The new positive primitive should be the closer of the two to the negative redex.

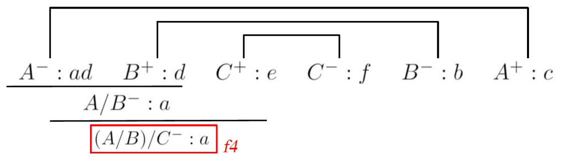
We then combine the positive primitive and the redex to form a new complex, negative category. At the same time, we give this new complex category the old semantic label of the redex, and assign the string consisting of this old label followed by the fresh variable on the new positive primitive as the new label of the redex.

In our example, suppose we elect to add the new primitives to the right of $f1$. Then we have:

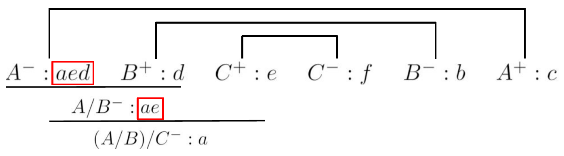


We add a link to join $f2$ and $f3$, and create a new formula $f4 = (A/B)/C^- : a$. In different words, we combine $f1$ and $f2$ into a larger term tree with the root, $f4$.





Then we replace the label on $f1$ with ae. We also replace a with ae in all of the descendants of $f1$'s term tree, if any.



We could have instead added $f3 = C^- : f$ and $f2 = C^+ : e$ (in this order) on the *left* of $f1$, and joined $f2$ and $f1$ with $f4 = (C\backslash A)^- : a$.

Regardless of the choice of left or right, we must also modify the corresponding LC-graph:

1. Add new nodes e and f,
2. for the unique value of x for which there is already an edge (x, a), add a new edge (x, e), and
3. add the new edge (e, f).

These measures will retain the correctness criteria in the LC-graph because, in this phase, there are no lambda nodes yet. So the new sequent will be derivable in the Lambek calculus if the sequent that began this iteration was.

We can repeat these steps as many times as we like in the first phase, and then decide to stop. If we apply zero iterations of the first phase, we cannot apply any iterations of the second phase either, and we will be left with the sequent $p \rightarrow p$.

3.2 Adding Lambda Nodes

Like the first phase, the second phase consists of the iterative application of a sequence of steps that we may elect to apply any number of times.

Every step in the second phase will preserve the number of axiomatic primitives and links, but will add one more lambda node. The following definition is useful:

Definition 3.2 (Anterior node)**.** *An anterior node, a, in an LC-graph is a positive node for which there is no lambda node on the path that leads from the root to a apart from possibly a itself.*

Definition 3.3 (Terminal node)**.** *A terminal node, t, in an LC-graph is a node with an out-degree of 0 and in-degree of 1.*

In Figure 2, j and i are both lambda nodes, but only j is an anterior lambda node. a, c, e and g are all positive non-lambda nodes, but only a, the root, is anterior. b and d are terminal nodes. f and h are negative, but not terminal because of their in-degrees. Terminal nodes are always negative because of their out-degree. Because of the restriction on their in-degrees, they correspond exactly to the labels of the LHS categories of the sequent.

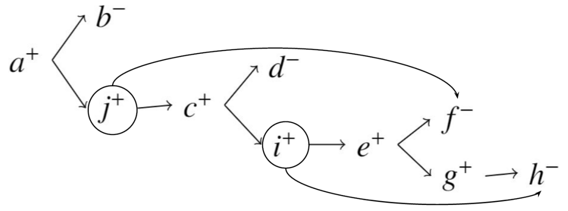


Figure 2: An example LC graph.

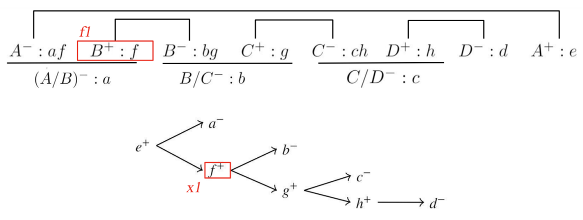


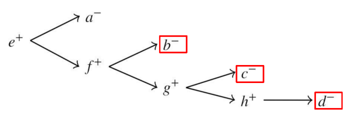
Figure 3: Running example for adding lambda nodes.

The redex, x_1, in each step of this phase will always be a positive node in the LC-graph that satisfies two other requirements:

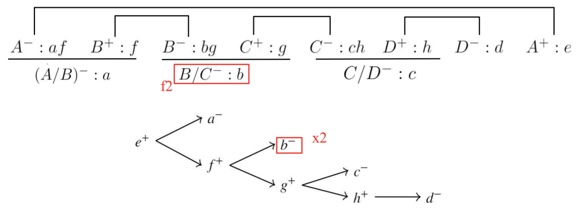
1. x_1 is anterior, and
2. x_1 has more than one terminal descendant.

Let $f1$ be the term or subterm labelled with x_1 (i.e. $f1 = X_1^+ : x_1$) in the proof net. We can do the following to the proof net (take Figure 3 as an example):

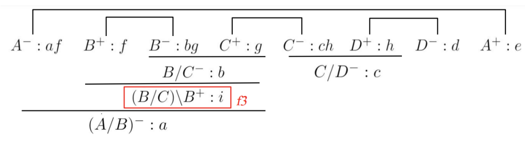
1. Let S be the set of terminal LC-graph nodes that are path-accessible from x_1.



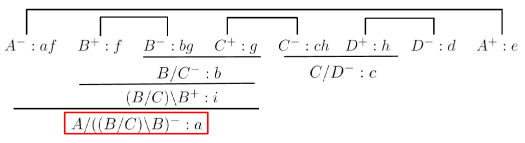
2. Pick $x_2 \in S$ as an outermost (either leftmost or rightmost) variable in S, arranged by the order of the elements in S within the current sequent. Let $f2$ be the (sub)term labelled with x_2 (i.e $f2 = X_2- : x_2$). In the running example, we choose x_2 to be the leftmost variable.



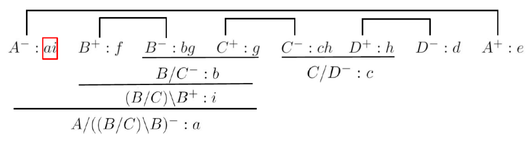
3. Create new formula $f3 = X_2\backslash X_1+ : x_3$, with $f1$ as the left child of $f3$ and $f2$ as the right child of $f3$. Insert $f3$ into the unfolding below $f1$.



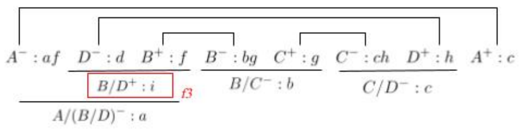
4. Update the categories in $f3$'s ancestors.



5. Replace x_1 with x_3 in all terms that are descendants of $f3$'s parent but not of $f3$.



In our example, we could have alternatively chosen $x_2 = d$, the rightmost variable in S, in which case $f3 = X_1/X_2+ : x_3$, with $f2$ as the left child $f1$ as the right child.



With either choice, we then modify the corresponding LC-graph:

1. Add x_3 as a new lambda node.

2. If x_1 is not the root, then for the unique y such that (y, x_1), replace this edge with (y, x_3). If x_1 is the root, do nothing.

3. Add edges (x_3, x_1) and (x_3, x_2).

Because x_1 was chosen to have more than one terminal descendant, Lemma 4.2 in the next section ensures that, after this step, the integrity criteria of the LC-graph are still satisfied. So the new sequent will be derivable in LCG.

3.3 End-to-end Example

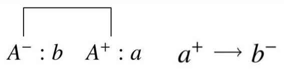
Figure 4 and Figure 5 depict the first and second phases, respectively, of a run of the generative process.

4 Coverage

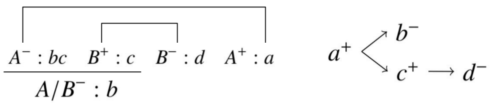
Theorem 4.1. *Every sequent that is derivable in the product-free Lambek calculus is derivable through the PLC generative process.*

All derivable sequents are witnessed by a valid proof net P paired with a corresponding LC-graph G. We will prove the above theorem in this section, by induction on the number of nodes in G. The base case, sequents of the form $A \models A$, where A is a primitive, are trivial to generate. In other instances, it suffices to show that every sequent, consisting of a valid proof net, P with corresponding LC-graph, G, can be derived from some other sequent (P', G') by running one more iteration of either the first phase or the second.

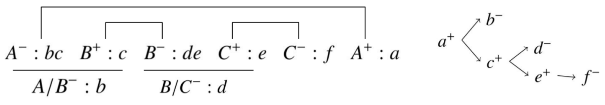
We will consider the following two cases: G contains lambda nodes, and G contains no lambda nodes.



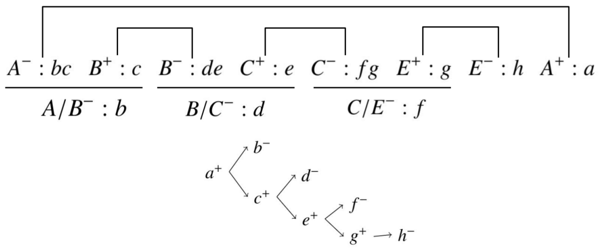
(a) initialization.



(b) first phase, first iteration.



(c) first phase, second iteration.



(d) first phase, third iteration.

Figure 4: An example run of non-lambda node generation.

4.1 LC-graph contains lambda nodes

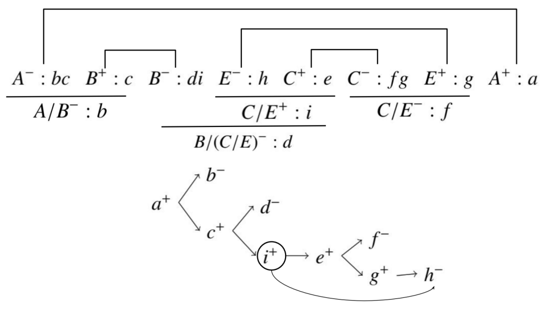
Lemma 4.2. *In any integral LC-graph, for any anterior lambda node v, there must be at least one terminal node v' such that $v \rightsquigarrow v'$.*

Proof. Every lambda node has two immediate descendants, one positive and one negative, and only these negative immediate descendants have indegrees of more than 1 (specifically, 2). If all negative nodes v' such that $v \rightsquigarrow v'$ had indegrees of 2, then they would all be negative immediate descendants of lambda nodes. Because v is anterior, those lambda nodes cannot be ancestors of v, and therefore must be v itself or descendants of v, and thus the x implied by **I(CT)** does not exist. □

Lemma 4.3. *If x, y are positive, w is negative, $x \rightsquigarrow w$ and $y \rightsquigarrow w$, then either $x \rightsquigarrow y$ or $y \rightsquigarrow x$.*

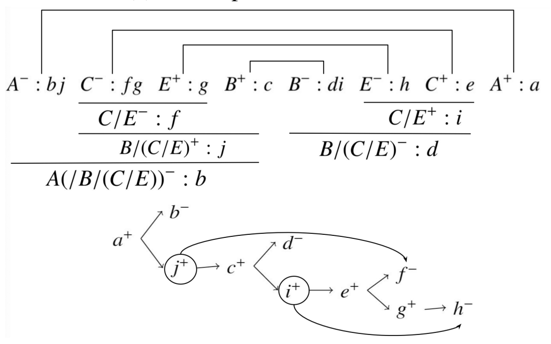
Proof. If w is the negative immediate descendant of a lambda node, z, then it is possible that one of x, y reaches w via z, and the other reaches it via the positive sibling of w, q, by **I(3)**. But $z \to q$, and so the result holds.

If w is not the negative immediate descendant of a lambda node, then its in-degree is 1 and its ancestors are all positive, with in-degrees of 1. So again the results holds. □



$A^- : bc$ $B^+ : c$ $B^- : di$ $E^- : h$ $C^+ : e$ $C^- : fg$ $E^+ : g$ $A^+ : a$
$A/B^- : b$
$C/E^+ : i$
$C/E^- : f$
$B/(C/E)^- : d$
a^+ b^- c^+ d^- i^+ e^+ f^- g^+ h^-

(a) second phase, first iteration.



$A^- : bj$ $C^- : fg$ $E^+ : g$ $B^+ : c$ $B^- : di$ $E^- : h$ $C^+ : e$ $A^+ : a$
$C/E^- : f$
$C/E^+ : i$
$B/(C/E)^+ : j$
$B/(C/E)^- : d$
$A(/B/(C/E))^- : b$
a^+ b^- j^+ c^+ d^- i^+ e^+ f^- g^+ h^-

(b) second phase, second iteration.

Figure 5: An example run of lambda node generation.

Lemma 4.4. *In any integral LC-graph, G, for any nodes $a, b, c \in G$, if $a \not\rightsquigarrow b$ and $b \not\rightsquigarrow a$, then there is no c such that $a \rightsquigarrow c$ and $b \rightsquigarrow c$.*

Proof. The only nodes with in-degrees greater than 1 (i.e., negative immediate descendants of lambda nodes) have out-degrees of 0. Thus if there were such a c, it would need to be a negative immediate descendant of some lambda node, d, itself. But even in this case, either $a \rightsquigarrow d \rightsquigarrow b$ or $b \rightsquigarrow d \rightsquigarrow a$ because of **I(3)**. □

Lemma 4.5. *For each term tree T in P, if $f2 = F2^{+/-} : y$ is a subterm of $f1 = F1^- : x$ in T such that $|y| = 1$ and there is no positive subterm of $f1$ of which $f2$ is a proper subterm, then for any ancestor, w, of x in G, $w \rightsquigarrow y$ in G.*

Proof. If $f1 = f2$, then $x = y$. Otherwise, the action of the negative unfolding rules on $f1$ guarantees that the primitive negative term label that reflects x will also reflect y, and so any node in G that leads to x through an axiom link also leads to y. We may consider the case when x is the negative immediate descendant of a lambda node, z, to be exceptional, because z points directly to x as a result of the action of the positive unfolding rules. But even in this case, by **I(3)**, $z \rightsquigarrow x$ through its positive immediate descendant, and thus also through an axiom link. □

Lemma 4.6. *For each term tree T in P, if $f2 = F2^{+/-} : y$ is a subterm of $f1 = F1^+ : x$ in T, where x is a lambda node and $|y| = 1$ (a variable), then $x \rightsquigarrow y$ in G.*

Proof. If $f1 = f2$, then $x = y$. Otherwise, because x is a lambda node, it has a positive immediate descendant, a, and negative immediate descendant, b, in G. b, in particular, corresponds to the negative immediate subterm of $f1$, $f3 = F3^- : b$. If $f2 = a$ or $f2 = b$, the lemma follows from the action of the positive unfolding rules.

Otherwise, we can obtain the result by induction on the number n of positive subterms of $f1$, q, for which $f2$ is a proper subterm of q. In the base case, $n = 1$, $f2$ is nested inside exclusively negative subterms of $f3$, and so by Lemma 4.5, $x \rightsquigarrow y$.

If $n > 1$, then let $f4 = F4^+ : z$ be the smallest positive subterm containing $f2$ other than possibly $f2$. By induction, $x \rightsquigarrow z$ in G. Because $f4$ contains $f2$, $F4$ is not primitive, and so z is a lambda node. Thus again by induction, $z \rightsquigarrow y$. □

Lemma 4.7. *For each term tree T in P, if $f2 = F2^{+/-} : y$ is a proper subterm of $f1 = F1^- : x$ in T, if $|y| = 1$ (a variable), then for any ancestor, w, of x in G, $w \rightsquigarrow y$ in G.*

Proof. Proof by induction on the number n of positive subterms of $f1$, q, such that $f2$ is a proper subterm of q. The base case, $n = 0$, follows by Lemma 4.5.

Otherwise, $n > 0$. Again, let $f3 = F3^+ : z$ be the smallest positive subterm of T containing $f2$ except possibly $f2$. By induction, every ancestor of x, $w \rightsquigarrow z$ in G. Because $f3$ contains $f2$, $F3$ is not primitive, and so z is a lambda node. Thus the result follows by Lemma 4.6. □

Lemma 4.8. *For every sequent with a primitive right-hand side $p^+ : r$, the axiom link emanating from $p^+ : r$ is incident to the axiomatic reflection of the label of some left-hand side term, i.e., $r \rightarrow b$ in G, for some left-hand side term, $B^- : b$.*

Proof. The axiom link must be incident to a negative-polarity primitive in the unfolding, which is either labelled with a string that includes the label of some left-hand side term, or with a string that includes the negative immediate descendant of some lambda node. In the latter case, the negative immediate descendant would be cut off from its positive immediate descendant sibling, in violation of **I(3)**, because p is primitive (and so there are no paths that would transit to b via r). □

Lemma 4.9. *Let X be the set of all axiomatic formulae of P, v be an anterior lambda node in G, and $D(v)$ be the set of terminal nodes w in G for which $v \rightsquigarrow w$ in G. Given a set of nodes, V, let $Q(V) = X \cap \{f \mid$ there exists a formula $f1$ with a label in V and f as one of its subterms in some term tree$\}$. Divide X into two subsets: $X1 = Q(\{v\}) \cup Q(D(v))$ and $X2 = X \backslash X1$. There is no axiomatic link connecting one formula in $X1$ with another one in $X2$.*

Proof. If v is the root of G, then $X1 = X$ and $X2 = \phi$. Otherwise, suppose there is such a link connecting $x1 \in X1$ and $x2 \in X2$. Let $A^{-/+} : a$ be the root of the term tree that contains $x2$.

Case 1: a is positive. A must be a primitive, otherwise, a would be a lambda node, and thus the only anterior node in G would be a itself. So $x2 = A^+ : a$. By Lemma 4.8, there exists a negative node b that labels the root of some term tree in G such that $a \rightarrow b$. Let $R^- : r$ be the root of the term tree that contains $x1$. If $b = r$, then by Lemma 4.7, b is the label of both $x1$ and its witness, $f1$. $b \neq v$ (wrong polarity), and so $b \in D(v)$, $v \rightsquigarrow b$, and because b is terminal, $v \rightsquigarrow a$, which would mean that G has a cycle. Thus $b \neq r$, but then b labels the root of a different term tree, and so $x2 = a \twoheadrightarrow x1$ after all. But a is the root of G, and so $x1 \twoheadrightarrow a = x2$ either.

Case 2: a is negative, and so it labels a left-hand-side term in the sequent, and thus it is terminal. So there exists a unique node b such that $b \rightarrow a$ in G. Because a dominates $x2 \in X2$, $v \not\rightsquigarrow a$, and so $v \not\rightsquigarrow b$. Furthermore, every node in the label of $x1$ is accessible from v. If $x1 \in Q(\{v\})$, then this follows from Lemma 4.6. If $x1 \in Q(D(v))$, then it follows from Lemma 4.7 and the definition of $D(v)$.

Case 2a: $b \not\rightsquigarrow v$ either. By Lemma 4.4, there is no node v' in G such that $v \rightsquigarrow v'$ and $b \rightsquigarrow v'$. A link between $x1$ and $x2$ will guarantee that there is such a v', however, because, by Lemma 4.7, every node in the label of $x2$ is accessible from b.

Case 2b: $b \rightsquigarrow v$. First, we begin by noting that $x2$ cannot simultaneously be negative and contain a in its label, or else the label of $x1$ is a single node that points to a, and hence $v \rightsquigarrow a$. So there must be a positive $C^+ : c$ subterm of $A^- : a$ which in turn contains $x2$ as a subterm. Choose the highest such subterm. Then because c is in the negative reflection of a, $b \rightarrow c$. Furthermore, either c is the label of $x2$ or, by Lemma 4.6, every node of $x2$ is accessible from c.

Let w be the negative member of $\{x1, x2\}$ and the target of the link between them. The nodes labelling w are accessible from both v (via $x1$) and c (via $x2$). By Lemma 4.3, either $v \rightsquigarrow c$ or $c \rightsquigarrow v$.

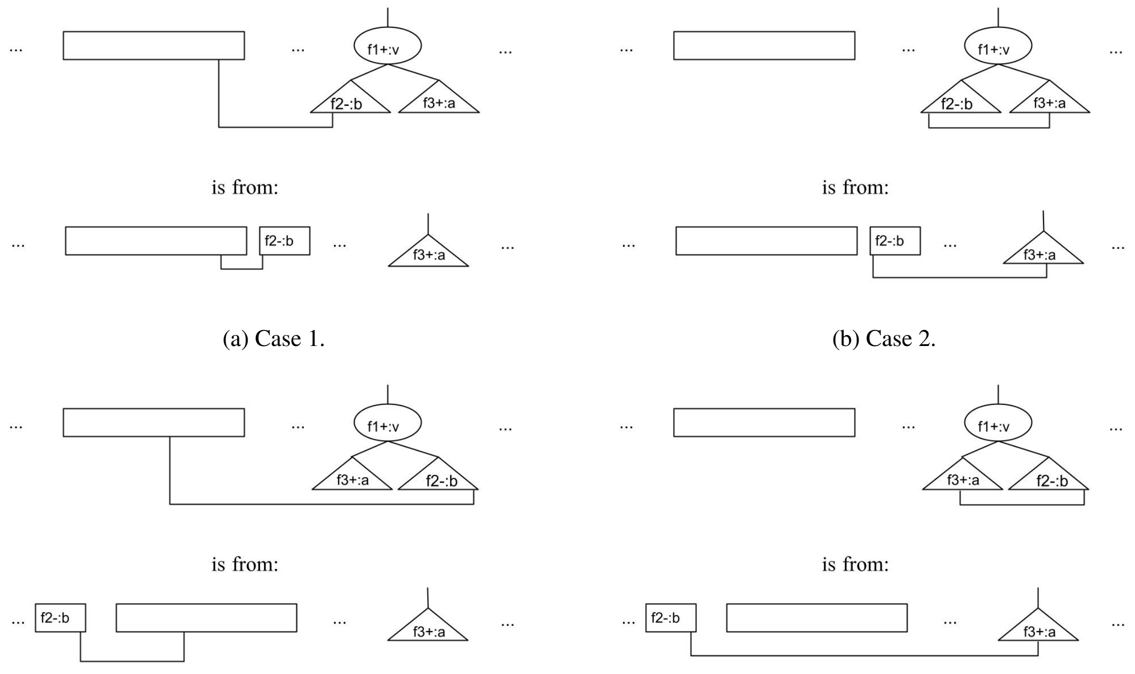
If $v \rightsquigarrow c$, then recall that both $b \rightarrow c$ and, by assumption, $b \rightsquigarrow v$, which means either $v = b$, which contradicts $v \not\rightsquigarrow a$, or $v = c$, which contradicts $x2 \in X2$. If $c \rightsquigarrow v$, then because v is anterior, c is the label of $x2$, which is positive, and therefore the source of the link between $x1$ and $x2$. As a result, the nodes of $x1$ are accessible from v, and immediately accessible from $x2$. Similar to the proof of Lemma 4.3, This might happen if $x1$ is the negative immediate descendant of some lambda node, z, but now we know that $x1$ and $x2$ are connected by an axiom link, and so $v \rightsquigarrow z$ and, by **I(3)**, $z \rightsquigarrow x2 = c$. Thus $v = c$, which is again a contradiction. □

Lemma 4.10. *For every anterior lambda node v in G, the LHS terms labelled by each of v's accessible terminal nodes must be contiguous in the sequent.*

Proof. This follows from Lemma 4.9. The terms that are labelled by each of the $D(v)$ are not connected by axiom links to the other terms, with the exception of the term that contains v itself, and even then only to the subterm rooted at v. If some of these other terms were interspersed among the former, then either the linkage would not be half-planar, or the underlying LC-graph would not be connected. Even the term that contains v itself cannot appear between two terms labelled by members of $D(v)$, because either v itself, which is positive, is the label of the term, and so must be the RHS category of the sequent, or it is not, in which case part of that term resides in $X2$, leading to the same contradiction. □

Theorem 4.11. *If G contains at least one lambda node, then (P, G) could have been generated in PLC from a proof net with any arbitrary anterior lambda node removed.*

Proof. Suppose v is the variable of the anterior lambda node in G, b is its negative immediate descendant in G and a, its positive immediate descendant.



(a) Case 1. (b) Case 2.

(c) Case 3. (d) Case 4.

Figure 6: Four cases of the respective ordering of $f2$, $f3$ and the contiguous subsequence of terms labelled with accessible terminal nodes, as given by Lemma 4.10 (depicted with a rectangle).

Let $f1$ be the formula labeled with v, $f2$ be the formula labeled with b and $f3$ be the formula labeled with a. In the parlance of Lemma 4.9, $X1$ consists of $f2$, $f3$, and the subsequence of terms labelled by the accessible terminal nodes of v. By Lemma 4.10, this subsequence is contiguous. By Lemma 4.9, the axiom links that have one side incident to any of these have both sides incident to these. There are therefore $2 \times 2 \times 2 = 8$ possible combinations: (1) whether $f2$ occurs to the left or right of $f3$, (2) whether the leftmost (resp. rightmost) axiomatic reflection of $f2$ connects to $f3$ or to a reflection within the contiguous subsequence and (3) whether the contiguous subsequence occurs to the left or right of $f1$. For brevity, Figure 6 shows the four possible cases in which we choose "left" for (3) — the other four are the symmetric duals of these.

In each case, such a derivation exists through a single iteration of the second phase of the generative process, as shown in Figure 6.

Note: the contiguous subsequence is, in fact, the left-hand side of the subsequent that derives $f1$. □

4.2 LC-graph does not have lambda nodes

Lemma 4.12. *For an arbitrary valid proof net P with its corresponding LC-graph G, if G does not contain any lambda node, then the sequent must be one of:*

$$A \models A$$

or

$$..., \phi/A, A... \models ...$$

or

$$..., A, A\backslash\phi.... \models ...$$

where A is a primitive and ϕ is any category.

Proof. By structural induction on the proof of the sequent. We will refer to A as a *reduction point* in the sequent. □

Theorem 4.13. *If G has at least three nodes and contains no lambda nodes, then (P, G) could have been generated in PLC from a proof net with any arbitrary reduction point removed.*

Proof. If G contains only two nodes, then the sequent must be $A \models A$. According to Lemma 4.12, in the second case, a reduction point exists, and corresponds to the result of a non-lambda-node addition step in PLC, as shown in Figure 7. Note that the term trees involving the reduction point must be adjacent in the sequent.

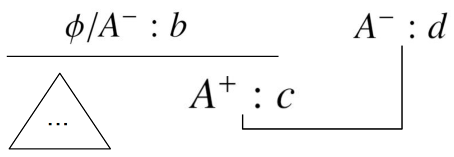


Figure 7: Formula f, f_1 and f_2 in proof net.

The third case is symmetric to the second. □

Corollary 4.14. *Any proof net and its corresponding LC-graph can be generated by starting with $p \rightarrow p$, followed by a sequence of steps of adding non lambda nodes then followed by a sequence of steps of adding lambda nodes.*

Proof. by Thm 4.11 and Thm 4.13, we can always put steps of adding non lambda nodes before the steps of adding lambda nodes. □

5 Adding Probability

Maximum likelihood parameter estimation can likewise be accomplished by separately parameterizing the two phases.

5.1 Phase 1: Adding Non-lambda Nodes

The first phase iteratively picks a category A and expands it to either $A/P \cdot P$ or $P \cdot P\backslash A$ for some primitive P. So for all primitives P, we must assign a probability to these two expansion schemes, as well as to $A \rightarrow A$, which means that A has been skipped with no expansion. The estimates are then simply the relative frequencies with which these rules have been chosen in this phase.

Below is a tiny example corpus with only two proof nets and one primitive:

$p1 : \; N1/N2 \quad N3 \quad \models N4$
axiom links : $(1,4),(2,3)$
$p2 : \; N1/N2 \quad N3 \quad N4\backslash N5 \quad \models N6$
axiom links : $(3,4),(2,5),(1,6)$

Here, the primitive category is N, and the numbers index the instances of it within the sequent.

For $p1$, we start from $N1 \models N4$, and proceed with these steps:

1. $N1 \rightarrow N1/N2 \quad N3$
2. $N1/N2 \rightarrow N1/N2$ (no expansion)
3. $N3 \rightarrow N3$ (no expansion)

For $p2$, we start from $N1 \models N6$, and follow with these steps:

1. $N1 \rightarrow N1/N2 \quad N5$
2. $N5 \rightarrow N3 \quad N4\backslash N5$
3. $N1/N2 \rightarrow N1/N2$ (no expansion)
4. $N3 \rightarrow N3$ (no expansion)
5. $N4\backslash N5 \rightarrow N4\backslash N5$ (no expansion)

Between the two derivations, there are a total of 8 steps, and so we can conclude that:

1. $P(A \rightarrow A/N \quad N) = \frac{2}{8}$
2. $P(A \rightarrow A) = \frac{5}{8}$
3. $P(A \rightarrow N \quad N\backslash A) = \frac{1}{8}$

Using this distribution, we know that:

$$\begin{aligned} P(p1) &= \tfrac{2}{8} \times \tfrac{5}{8} \times \tfrac{5}{8} = \tfrac{25}{256} \\ P(p2) &= \tfrac{2}{8} \times \tfrac{1}{8} \times \tfrac{5}{8} \times \tfrac{5}{8} \times \tfrac{5}{8} = \tfrac{125}{16384}. \end{aligned}$$

5.2 Phase 2: Adding Lambda Nodes

Once every category has been touched with a decision not to expand, it is time to start adding lambda nodes in the second phase. Let us modify the above corpus so that it requires a second phase, and yet can be derived through an identical sequence of steps as in the first phase:

$p1 : \; N1/N2 \quad \models N4/N3$
axiom links : $(1,4),(2,3)$
$p2 : \; N1/(N3\backslash N2) \quad N4\backslash N5 \quad \models N6$
axiom links : $(3,4),(2,5),(1,6)$

Both $p1$ and $p2$ contain one lambda node. During the second phase, we must first determine the place to add a lambda node, and then select either $(\backslash R)$ or $(/R)$. $p1$ applies $(/R)$ once and $p2$ applies $(\backslash R)$ once. So, for this corpus, $P(/R) = P(\backslash R) = \frac{1}{1+1} = \frac{1}{2}$.

The only remaining problem is how to determine where to add lambda nodes. Once this problem is solved, then:

$$\begin{aligned} P(p1) &= \tfrac{25}{256} \times P(\textit{pick positions in } p1) \times P(/R) \\ P(p2) &= \tfrac{125}{16384} \times P(\textit{pick positions in } p2) \times P(\backslash R) \end{aligned}$$

Now we will present our method for selecting places to add lambda nodes.

Definition 5.1 (Breakable)**.** *A node v in an integral LC-graph is **breakable** iff we can first:*

1. *if v is not the root, add a lambda node a between the edge $u \to v$, or*
2. *if v is the root, add a lambda node a such that $a \to v$,*

then pick a terminal node w such that $v \rightsquigarrow w$, then add (a, w), and the integrity of the LC-graph is preserved.

If v can be picked as $x1$ in Section 3.2, then it is breakable, but the inverse may not be true.

Lemma 5.1. *An LC-graph satisfies I(CT) iff for every lambda node v:*

$$\begin{aligned} &|\{u \mid u \text{ lambda node } \& \ v \rightsquigarrow u\}| \\ &\quad < \ |\{w \mid w \text{ negative } \& \ v \rightsquigarrow w\}| \,. \end{aligned}$$

Proof. Every lambda node u points to its positive immediate descendant. So:

$$\begin{aligned} &|\{u \mid u \text{ lambda node } \& \ v \rightsquigarrow u\}| \\ &\quad \leq \ |\{w \mid w \text{ negative } \& \ v \rightsquigarrow w\}| \,. \end{aligned}$$

And the node x asserted by I(CT) is a negative node that is not accessible from any lambda node. So the inequality is strict. □

Lemma 5.2. *A node v is breakable iff:*

$$\begin{aligned} &|\{u \mid u \text{ lambda node } \& \ v \rightsquigarrow u) + 1\}| \\ &\quad < \ |\{w \mid w \text{ negative } \& \ v \rightsquigarrow w\}| \,. \end{aligned}$$

Proof. This readily follows from Lemma 5.1. □

Theorem 5.3. *If a node v is breakable, then picking it as $x1$ during Section 3.2 will not change the breakability of any node u such that $v \not\rightsquigarrow u$.*

Proof. If $u \not\rightsquigarrow v$, the constraints of breakability in Lemma 5.2 will not be changed. If $u \rightsquigarrow v$, then the constraints of Lemma 5.2 must have been satisfied both before and after adding the lambda node. So in both cases, the breakability of u will not change. □

So in the second phase, we will add lambda nodes from bottom to top. So we first gather all breakable nodes into a set denoted as S. During each iteration, we find the subset $U \subseteq S$ with maximum depth. We then either apply Section 3.2 by picking nodes in U or reject U as candidates and jump to the next lower lower depth. Thus, we need a probability p that a given breakable node will be selected. Eventually S will be empty. Here is the algorithm:

Algorithm 1: Redex selection during the second phase.

```
   Result: proofnet and LC-graph
 1 $S$ = all breakable nodes;
 2 while $S$ is not empty do
 3     $MaxDepth$ = max depth of node in $S$;
 4     $U = \{u | u \in S, u's\ depth = MaxDepth\}$;
 5     $S = S \setminus U$;
 6     while $U$ is not empty do
 7         $u$ = pick node from $U$ uniformly;
 8         $flag$ = Sample from Bernoulli($p$);
 9         if $flag$==1 then
10             let $u$ be $x1$ and add lambda node $a$;
11             $U = U - u$;
12             if $a$ is breakable then
13                 $U = U + a$;
14             end
15         else
16             $U = U - u$;
17         end
18     end
19 end
```

The set U is finite, so lines 6–18 will eventually terminate. During each iteration of line 2–19, the size of S strictly decreases, so the entire algorithm eventually terminates.

At line 4, $\nexists u, v \in U$ such that $u \neq v$ and $u \rightsquigarrow v$. During lines 6–18, denote U^i as the U at line 6 and at iteration i. For all $u \in U^i$, all $v \in U^{i+1}, u \not\rightsquigarrow v$. Theorem 5.3 ensures that lines 9–17 will not change the breakability of the nodes in U^{i+1}. Also, if we denote U^i as the U in line 4, the nodes in U^{i+1} have smaller depth, so for all $u \in U$, all $v \in U^{i+1}, u \not\rightsquigarrow v$. Theorem 5.3 also ensures lines 3–18 will not change the breakability of the nodes in U^{i+1}.

Returning to our tiny corpus, during the second phase, $p1$ traverses lines 10–14 once and line 16 zero times, while $p2$ traverses lines 10–14 once and line 16 once. Therefore $p=\frac{2}{3}$. Thus:

$$\begin{aligned} P(p1) &= \tfrac{25}{256} \times \tfrac{2}{3} \times \tfrac{1}{2} = \tfrac{25}{768} \\ P(p2) &= \tfrac{125}{16384} \times \tfrac{2}{3} \times \tfrac{1}{3} \times \tfrac{1}{2} = \tfrac{125}{147456}. \end{aligned}$$

6 Experimental Investigation

6.1 Dataset

We trained and tested our probabilistic generative model on LCGbank (Fowler, 2016), a conversion of CCGbank (Hockenmaier and Steedman, 2007) to LCG. CCGbank is in turn a conversion based upon the Penn TreeBank (Marcus et al., 1993). We use LCGbank section 23, which contains 2416 proof nets as the test set. For the training set, we use sections 1-22 and 24 which contain 44870 proof nets. LCGbank uses the 5 primitives, ”S”, ”NP”, ”N”, ”conj” and ”PP”. The number of tokens is slightly different between CCGbank and PTB for the following two reasons:

- Some tokens like punctuation do not have categories in CCGbank. So we ignore those tokens in both LCGbank and PTB.

- Tokens like ”interest-rate” count as one token in CCGbank but count as three in PTB.

6.2 Comparison with Probabilistic Context Free Grammar

Since our model uses MLE, we also estimate the parameters for a PCFG using MLE on the PTB (using the same sections as the training and test set). Also, we exclude the lexica for both the PCFG and PLC. For the PCFG, the task is merely to generate sequences of POS tags, and for PLC, it is to generate sequents with the right-hand side of S.

Our model can generate every sequence in the test set because all primitives in the test set had been seen in the training set. Thus it can also assign a positive probability to every proof net in the test set. Some production rules in the PTB test set, however, never appear in the training set, and so PCFG fails to assign a non-zero probability to some sentences in the test set. Table1 shows that 260 sentences have zero probability in the resulting PCFG. On the other hand, the majority of

	PCFG	PLC
$P \geq 0$	2144	2416
$P = 0$	272	0

Table 1: Number of sentences receiving positive and zero probabilities.

sentences assigned a non-zero probability by both PLC and PCFG are assigned a lower probability by PLC than by PCFG. Table2 shows the number of cases (excluding zero-probability cases) that receive larger and smaller probabilities using PCFG. Table 3 shows the log probability of the entire test set (sum of log probability of each sentence) divided by the number of tokens. This evidence is consistent with the conclusion that PLC spreads probabilities more evenly across the seen and unseen category sequences induced by the set of primitives, at the expense of the data likelihood of the corpus.

Positive	Negative
1964	180

Table 2: Number of sentences for which $p(PCFG) - p(PLC)$ is positive and negative, respectively.

PCFG	PLC
-2.95	-4.06

Table 3: Log probability of test set normalized by the number of tokens.

7 Conclusion and Future work

In this work, we have presented a probabilistic generative model for sequent derivability in the Lambek calculus. Any sequent that is derivable in the Lambek Calculus can be generated by our model. We also compared PLC with PCFG using MLE, both trained and tested on comparable corpora. The results show a trade-off to using PLC, in which the probabilities are more evenly distributed.

This probabilistic model may be used to parse with Lambek Grammar, although further investigation of early stopping when categories expand outside the coverage of the lexicon is still needed.

Another advantage of numerically parametrizing a grammar is to improve the speed of parsing, typically by forcing early failures. This is a direction that we have not yet pursued.

References

S. Bangalore and A. Joshi, editors. 2010. *Supertagging: Using Complex Lexical Descriptions in Natural Language Processing*. MIT Press.

G. Bonfante and P. de Groote. 2004. Stochastic lambek categorial grammars. *Electronic Notes in Theoretical Computer Science*, 53:34–40.

T. A. Fowler. 2016. *Lambek Categorial Grammars for Practical Parsing*. Ph.D. thesis, University of Toronto.

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

T. Huang and K.S. Fu. 1971. On stochastic context-free languages. *Information Sciences*, 3(3):201–224.

S. Kübler, R. McDonald, and J. Nivre. 2009. *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool.

J. Lambek. 1958. The mathematics of sentence structure. *The American Mathematical Monthly*, 65(3):154–170.

K. Lari and S.J. Young. 1991. Applications of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech & Language*, 5(3):237–257.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

M. Osborne and T. Briscoe. 1997. Learning stochastic categorial grammars. In *Proc. CoNLL*, pages 80–87.

G. Penn. 2004. A graph-theoretic approach to sequent derivability in the lambek calculus. *Electronic Notes in Theoretical Computer Science*, 53:274–295.

M. Pentus. 1993. Lambek grammars are context free. In *[1993] Proceedings Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 429–433.

M. Pentus. 1997. Product-free lambek calculus and context-free grammars. *J. Symb. Log.*, 62:648–660.

D. Roorda. 1991. *Resource Logics: Proof-Theoretical Investigations*. Ph.D. thesis, Universiteit van Amsterdam.