On the Connection between Context-Free Grammars and Lambek Categorial Grammars

(Extended Abstract)

Gerald Penn[*]
Computational Linguistics Program,
Carnegie Mellon University
and
OTS Utrecht
gpenn+@cmu.edu

Overview

A method is described for proving the weak equivalence theorem between natural-deduction style Lambek Categorial Grammmars (LCG) and Context-Free Grammars (CFG). The method is achieved using a simple representation for derivations from the associative, product-free Lambek Calculus in the LF meta-language [10], which can be adapted to the representation of other substructural logics in LF. The representation achieves its high level of abstraction, and thus its simplicity, in two ways. First, it refines several attempts to adapt chart parsing to Lambek grammars into a data structure that allows LCG proof objects to be represented as simple LF objects at the meta-level while preventing unwanted applications of weakening and exchange, to which the Lambek Calculus is sensitive. Secondly, it exploits LF's ability to represent parametric deductions either as higher-order objects or as objects evaluated in the context of meta-level variables representing their parameters. One direction of this proof has been implemented in Elf [22], a logic programming language based on the LF formalism, thus providing a means to transform automatically between CFG derivations and derivations in the LCG's induced by those CFG's. Using this representation to implement a specification for LCG deductions and one direction of the weak equivalence theorem has revealed a formal way to represent the distribution of phrase structure rule information that may lead to a more constrained proof of the other direction of the weak equivalence theorem, and ultimately, to a solution to the parsing complexity problem for Categorial Grammar.

1 Introduction

Substructural logics are becoming increasingly more important in computational linguistics as a means of formalising linguistic phenomena and their interaction. The substructural logic which will be of primary concern here is called the *associative, product-free Lambek Calculus.* Given a set of *basic categories*, **BasCat**, we define a *category system*, **Cat** to be the smallest set containing **BasCat**, $\{x/y \mid x, y{\in}\mathbf{Cat}\}$, and $\{x\backslash y \mid x, y{\in}\mathbf{Cat}\}$. The *associative, product-free Lambek Calculus* over a category system, **Cat**, is then defined by the following deductive schemata[1]:

$$\frac{a/b \qquad b}{a}\,/\mathsf{e} \qquad\qquad \frac{b \qquad a\backslash b}{a}\,\backslash\mathsf{e}$$

$$\begin{array}{c} \overline{b}^{\,u} \\ \vdots\ \vdots \\ a \\ \hline a/b \end{array}\,/\mathsf{i}^{\mathsf{u}} \qquad\qquad \begin{array}{c} \overline{b}^{\,u} \\ \vdots\ \vdots \\ a \\ \hline a\backslash b \end{array}\,\backslash\mathsf{i}^{\mathsf{u}}$$

[0] The author would like to thank Frank Pfenning for his very helpful advice at every stage in this project. This material is based upon work supported under a National Science Foundation Graduate Research Fellowship.

[1] I shall use the so-called *Steedman notation* in this paper. So a category $sent\backslash np/np$ takes a category np on its right to form a category $sent\backslash np$, which then takes a category np on its left to form a category $sent$.

where $a, b \in \mathbf{Cat}$. The rules of this calculus are sensitive to the order of the premises, and all of the premises provided to a derivation must be consumed. So in the rule for $/\mathsf{i}^{\mathsf{u}}$, for example, the hypothetical deduction, u, must be consumed, and must be the right-most deduction used in the derivation of a.

A *Lambek Categorial Grammar*, consists of a set of basic categories, **BasCat**, which will be treated as understood in this paper, and a triple $\langle W, f, d\rangle$, where W is a set of words, $f : W \rightarrow P(\mathbf{Cat})$ is an initial assignment function, and d is the distinguished basic category. Given an LCG, G, the *language of* G, $L(G)$, is the set of all strings of words, $w_1 \ldots w_n$, such that, in the associative, product-free Lambek Calculus:

$$\begin{array}{c} \dfrac{}{x_1}\,\mathsf{u}_1 \quad \cdots \quad \dfrac{}{x_n}\,\mathsf{u}_n \\ \vdots \\ d \end{array}$$

where $x_i \in f(w_i)$, and d is the distinguished basic category of G.

Why should computational linguists be concerned with such a logic? Just as Curry and Howard were able to associate proofs of propositions with the assignment of types to objects in the realm of logic and programming languages, so Pereira and others have associated proofs of propositions with parses that assign grammatical types of *constituency*, such as noun phrase (np) or sentence ($sent$), to strings in natural language. This was, in fact, one of the guiding principles behind the creation of Prolog, which was originally designed for natural language processing [21]. To find whether *John likes Mary* is a grammatical sentence in English in a natural-deduction style, we can try to find a proof that the sentence is in $L(G)$, where G is the putative LCG for English, with a distinguished category, $sent$, corresponding to sentence-hood:

$$\dfrac{\dfrac{}{np}\,\mathsf{u}_1 \qquad \dfrac{\dfrac{}{sent\backslash np/np}\,\mathsf{u}_2 \quad \dfrac{}{np}\,\mathsf{u}_3}{sent\backslash np}\,/\mathsf{e}}{sent}\,\backslash\mathsf{e}$$

This deduction is *parametric* in deductions, u_1, u_2, and u_3, for the categories of *John*, *likes*, and *Mary*, respectively. In other words, given any deductions of these categories for u_1, u_2, and u_3, this provides a deduction of $sent$. Notice that the parametric deductions are for categories like np, not words like *John*. Thus this proof is valid given any np in that position, whether it is *John*, *The boy*, or *The tall, well-groomed gentleman with the umbrella.*

Historically, the category assignment of an LCG has only had to weakly correspond to linguistic sentiments about such constituent types — as long as the same strings are assigned the category $sent$ as the ones a linguist believes to be grammatical sentences, it did not matter what categories various substrings were said to have. This led to LCG's which massively overgenerated strings, when compared to the natural languages they were intended to model. More recently, some linguists have augmented this classical calculus with various modalities to represent some of the more complex linguistic processes in a more constrained fashion [4, 11, 17, 18].

On the more formal side of computational linguistics, the weak equivalence of LCG's and CFG's has now been properly established [7, 19], renewing concern about efficient parsing algorithms for the Lambek calculus and about the theoretical complexity of parsing for it and related substructural logics [1, 14, 15]. Only recently, moreover, the ramifications of the Curry-Howard isomorphism for LCG's and linear logic have become clear, relating Categorial Grammar proof objects, for example, to a fragment of the simply typed lambda calculus.

Both of these perspectives represent the growing need for a programming language that can easily represent and manipulate proof objects in these calculi, including higher-order objects (which Prolog cannot do). Elf excels at just this sort of manipulation; but its structural rules grant it an inferential capacity that is too powerful to express deductions in these calculi straightforwardly as deductions in the meta-language. For example, if we want to find a derivation of the sentence, *John is eating the broccoli*, we do not want a derivation of *The broccoli is eating John* (by exchange), or of *John is eating* (by weakening). One could use a logic programming language based on linear logic, such as LOLLI [13], to avoid weakening[2],but exchange would still be a problem, and it would not be possible to represent higher-order proof objects as elegantly in LOLLI

[2]Hodas & Miller [13], for example, uses linear logic in a DCG for gapping constructions to handle subject-object assymetries and the constraint that gaps, once introduced, must be used.

as in Elf. There is a system under development, LLF [6], a logical framework based on linear logic, which would allow the same elegance of representation, but also enforce the proper consumption of resources at the same time. The representation and proof presented in this paper might possibly be improved using LLF, once it is released.

This paper will present a natural way to express deductions of a substructural logic, in particular the classical associative Lambek Calculus, in the LF formalism, by means of generalizing the data structure used in *chart-parsing*, a technique for efficiently parsing with ambiguous CFG's. This method will then be used in a formal representation of one direction of the proof of weak equivalence between CFG's, and LCG's, which provides a way to transform derivations in a CFG to derivations in its induced LCG, and vice versa. In this paper, I will simply provide the Elf code for the LF types and objects that are defined.

It should be emphasized, however, that while the representation being used draws its inspiration from a data structure used in parsing, and while the implementation of the aforementioned theorem can transform proof objects that would be typical outputs of a parser, the present work does not include the implementation of a parsing strategy. In fact, it remains an open question whether derivations in the classical associative Lambek Calculus can be found in polynomial time with respect to the length of the derived string.

2 Chart-Parsing

Chart-parsing refers to a collection of parsing strategies that employ memoization to trim the complexity of parsing with ambiguous CFG's [8], although I will not be concerned with complexity issues here. For every string of length n that is parsed, a graph, or chart, is constructed with $n+1$ nodes, usually numbered from 0 to n, and represented in a line. Every pair of adjacent nodes $(m-1, m)$ is connected with an edge labelled by the mth terminal of the string. In general, an edge in the chart is labelled with either a terminal or non-terminal from the CFG in question; and an edge spanning from node m_0 to m_1 with label X means that the substring consisting of the m_0+1st through the m_1th terminals can be rewritten from non-terminal or terminal X. The chart is filled with edges, using the rules of the CFG, by one of a number of control strategies that have been proposed; and the string can be said to be recognized by the CFG if there is an edge that spans from node 0 to node n (a *spanning edge*) whose label is the distinguished non-terminal of the grammar (Figure 1).

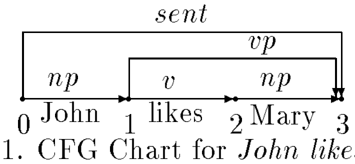

Figure 1. CFG Chart for *John likes Mary*

Each edge with a non-terminal label, which, thus, corresponds to the left-hand-side of a CFG rule application, can also be labelled with pointers to the edges corresponding to the elements of the right-hand-side of that CFG rule. Then, once the chart is filled, a parse tree (or an equivalent sequence of rewritings) can be reconstructed from every distinguished spanning edge.

3 Representing LCG deductions

A similar representation can be given for LCG deductions. If it were not for the introduction rules, /i and \i, we could use exactly the same representation. For every lexical item, w_i, an edge between two adjacent nodes could be assigned, labelled with one of the categories from $f(w_i)$. Deductions resulting from an elimination rule would be represented by an edge spanning its adjacent edge premises. Introduction rules, however, have deductions as their premises which use hypothetical deductions. These hypothetical deductions cannot be represented as normal edges on the same chart like other deductions. From the standpoint of a parsing representation, there would be nothing to prevent the parser from using a deduction with the same nodes and category rather than the hypothetical one, or vice versa. In CFG chart parsing, if one identically-labelled edge is used rather than another, there is no difference. But edges corresponding to hypothetical deductions can only be used to form other edges in the scope in which they are defined, whereas their identical counterparts could have a different scope. What is required, therefore, is a level of abstraction that allows the chart, rather than the representational objects, to manage the scope.

Various modifications of chart-parsing have been proposed for LCG's. Koenig [16] augments the traditional chart-parser with meta-rules that "spawn" an entirely new chart for every application of an introduction rule. Hepple [12] proposed incorporating these many charts into a single "multi-dimensional chart" where each lexically motivated edge and each hypothetical edge governs a "primitive interval" which is related to other primitive intervals by a partial ordering (as opposed to the intervals of node pairs on a CFG chart, which are totally ordered). Edges resulting from elimination govern the union of their premises' intervals. Edges resulting from introduction govern the interval of their premise, minus the primitive interval of their hypothetical deduction. These orderings can then be used to eliminate scoping problems, since edges that were not supposed to use the primitive interval for hypothetical deductions will never be able to produce edges that span all and only the lexical primitive intervals.

The representation used here is essentially that same multi-dimensional chart, only edges are still referred to by a pair of nodes, as in CFG parsing, not by orderings of primitive intervals. This has the advantage that every edge, no matter how many "intervals" it spans, can be represented by three pieces of data: a left node, a right node, and a category. This representation will also be faithful to the multi-dimensional nature of the chart by representing hypothetical deductions as edges which are parametric in one of their nodes.

Let us consider, as an example, the representation of a deduction for the sentence, *John likes Mary* (Figure 2):

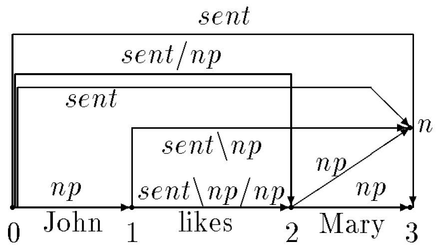

Figure 2. LCG Chart for *John likes Mary*

The node, n, which is drawn above the baseline where the others appear is a hypothetical node, and the edge which connects it, a hypothetical deduction, representing the np necessary to apply a $sent\backslash np/np$ category to its inner argument before its outer one. For this sentence, this is not necessary, of course; but if the sentence were *John likes and Bill hates Mary*, then it would be necessary in order to conjoin *John likes* and *Bill hates* (where *and* would be of category, $(sent/np)\backslash(sent/np)/(sent/np)$).

Notice that one can simply use a hypothetical node as a left node or a right node without worrying about overlapping sequences of intervals, as in Hepple's formulation, because, in Lambek Categorial Grammar, hypothetical deductions are always the right-most or left-most deductions in the scope in which they are defined; and, as a result, the hypothetical node will be either the left or right node of the premise to an introduction rule. Notice also that the use of numbered nodes prevents exchange or weakening, since elimination can only combine two edges where the left node of one is less than the right node of the other, and which agree on a third, intermediate node.

From the standpoint of representing these deductions in Elf, parametric deductions, such as the ones in the premises of the introduction rules, will, of course, be represented by higher-order objects. The variable over which those objects are abstracted represents the hypothetical deduction of an introduction rule. I will also want to use beta-reduction in the meta-language to apply these higher-order objects to other proof objects to obtain the representations of new proofs (the significance of which will be explained below). But the argument objects with which these variables must unify could represent edges of any category, and spanning any number of nodes. For this reason, representations of hypothetical deductions must be *parametric in their category* and *in one of their nodes* as well (the left node for `\i`, and the right node for `/i`).

An LCG deduction in LF uses the aforementioned three pieces of information, just like an edge in a parsing chart:

```
ded: nat -> nat -> syn -> type.
```

where `nat` is the declared type for natural numbers, and `syn` is the declared type for a category system.

The associative, product-free Lambek Calculus is represented by only functional proof objects, i.e. there are no axioms. Instead, a derivation of a string of words for an arbitrary LCG is represented by a deduction parametric in the categories of those words. Since we are using LF, those parametric deductions can be represented as parametric deductions in the meta-language. The representational objects provided here, however, do not represent those deductions as higher-order objects, but assume that the variables corresponding to the hypothetical deductions for the categories of words occur in every context in which these objects will be evaluated. One can also define an additional layer of representation in which the LCG derivation judgement is generalized over the LCG being used; and at this level, the categories for the words of a string, and the ordering of their baseline nodes can be introduced as hypothetical deductions in a manner similar to the hypothetical deductions of introduction rules.

The objects for the introduction rules, as stated above, will apply to objects that prove the category `B`, given a hypothetical deduction of an introduced category, `A`, to produce a deduction of `B\A` or `B/A`. They must also apply, however, to an object representing a proof that a natural number `L` is less than some `R`. Similarly, the objects for elimination rules must apply to objects that prove that a number `M` occurs strictly between two numbers `L` and `R`. The objects for `/i` and `\i` must also introduce a hypothetical deduction of a now rather unnatural natural number, `n`, along with a less-than deduction to link `n` to the regular nodes on the baseline of an LCG chart. In Elf, this is represented as:

```
/e: ded L R A
     <- between L R M
     <- ded L M (A / B)
     <- ded M R B.
\e: ded L R A
     <- between L R M
     <- ded L M B
     <- ded M R (A \ B).
/i: ded L R (B / A)
     <- less L R
     <- ({n:nat} less R n -> ded R n A
          -> ded L n B).
\i: ded L R (B \ A)
     <- less L R
     <- ({n:nat} less n L -> ded n L A
          -> ded n R B).
```

This representation is adequate for deductions in the associative, product-free Lambek calculus:

Theorem 1 (Adequacy of Representation) *Given categories,* $\mathrm{X}, \mathrm{X}_1, \ldots, \mathrm{X}_\mathrm{n}$*, there is a compositional bijection,* $\ulcorner\cdot\urcorner$*, between proofs of:*

$$\begin{array}{ccc} \overline{\mathrm{X}_1}\,\mathtt{u}_1 & \cdots & \overline{\mathrm{X}_\mathrm{n}}\,\mathtt{u}_n \\ & \mathcal{D} & \\ & \mathrm{X} & \end{array}$$

and objects $\mathrm{D} = \ulcorner\mathcal{D}\urcorner$*, such that:*

$$\Delta \vdash_{\mathrm{ND}^+} \mathrm{D} \Downarrow \mathtt{ded}\ \mathtt{m}_0\ \mathtt{m}_\mathtt{n}\ \ulcorner X \urcorner$$

where:

$$\begin{aligned}\Delta = {} & \mathtt{m}_0 : \mathtt{nat}, \ldots, \mathtt{m}_\mathtt{n} : \mathtt{nat}, \mathtt{l}_1 : \mathtt{less}\ \mathtt{m}_0\ \mathtt{m}_1, \ldots, \\ & \mathtt{l}_\mathtt{n} : \mathtt{less}\ \mathtt{m}_{\mathtt{n}-1}\ \mathtt{m}_\mathtt{n}, \mathtt{u}_1^+ : \mathtt{ded}\ \mathtt{m}_0\ \mathtt{m}_1\ \ulcorner X_1 \urcorner, \ldots, \\ & \mathtt{u}_\mathtt{n}^+ : \mathtt{ded}\ \mathtt{m}_{\mathtt{n}-1}\ \mathtt{m}_\mathtt{n}\ \ulcorner X_n \urcorner\end{aligned}$$

The bijection, $\ulcorner\cdot\urcorner$*, is compositional in the sense that, for proofs in such contexts,* Δ*, and* Δ'*:*

$$\begin{array}{ccc} \overline{\mathrm{X}_1}\,\mathtt{u}_1 & \cdots & \overline{\mathrm{X}_\mathrm{n}}\,\mathtt{u}_n \\ & \mathcal{D} & \\ & \mathrm{X} & \end{array} \quad \textit{and} \quad \begin{array}{ccc} \overline{\mathrm{X}'_1}\,\mathtt{u}'_1 & \cdots & \overline{\mathrm{X}'_\mathrm{j}}\,\mathtt{u}'_j \\ & \mathcal{D}' & \\ & \mathrm{X}_\mathrm{i} & \end{array}$$

$\ulcorner[\mathcal{D}'/\mathtt{u}_\mathtt{i}]\mathcal{D}\urcorner = [\mathtt{l}'/\mathtt{l}_\mathtt{i}][\ulcorner\mathcal{D}'\urcorner/\mathtt{u}_\mathtt{i}^+][\mathtt{m}'_\mathtt{j}/\mathtt{m}_\mathtt{i}][\mathtt{m}'_0/\mathtt{m}_{\mathtt{i}-1}]\ulcorner\mathcal{D}\urcorner$, *where* $\mathtt{m}'_0$ *and* $\mathtt{m}'_\mathtt{j}$ *are in* Δ'*, and* $\mathtt{l}' : \mathtt{less}\ \mathtt{m}'_0\ \mathtt{m}'_\mathtt{j}$ *is constructed from the* `less` *objects in* Δ'*.*

A similar deduction, `edge`, can be defined to build an adequate representation of CFG rewritings from a terminal or non-terminal symbol. One object, for example, is:

```
rule_app_unit: edge L R (n N)
           <- less L R
           <- rule N T nil
           <- edge L R (t T).
```

This object corresponds to a rewriting step that uses a unit rule, `N`→`T`, for non-terminal `N`, and terminal `T`.

4 Right-Atomicity

Before proceeding to the main equivalence theorem of this paper, it is necessary to make a few more useful definitions. The first set makes statements about the structure of categories; and the second, about the structure of deductions in the Lambek Calculus.

A category, C, is said to be *right-atomic* if either:

1. C is basic (sometimes called *atomic*), or
2. $C = B/A$, where A is atomic, and B is right-atomic.

If C is right-atomic by virtue of the second condition, C is said to be *strictly* right-atomic.

In Steedman notation, C is atomic if it has no slashes, and right-atomic if it only has forward slashes separating atomic arguments (since $A/B/C$ represents $(A/B)/C$, not $A/(B/C)$). The *head* of a right-atomic category, in Steedman notation, is the left-most atomic category (such as A in $A/B/C$). The head category is the category that a non-basic category produces once it has combined with all of its argument categories.

5 Normal Form for Lambek Calculus Deductions

In Figure 2, an introduction rule was used for the sake of illustration, although the derivation could have been constructed entirely from applications of elimination rules. In fact, there are an infinite number of derivations for every string which has at least one derivation in the Lambek Calculus, obtained by using superfluous instances of /i on the functor (left) side of /e, and of \i on the functor (right) side of \e, which "introduce" slashes that are already present.

Given the following *normalization scheme* (from Carpenter [5]):

$$\dfrac{\dfrac{\begin{array}{c} \mathsf{u}_1 \ \cdots \ \mathsf{u}_k \ \ \overline{}\,\mathsf{u}_{n+1} \\ B \\ \mathcal{D}_1 \\ A \end{array}}{A/B}\,/\mathsf{i}^{\mathsf{u}_{n+1}} \qquad \begin{array}{c} \mathsf{u}_{k+1} \cdots \mathsf{u}_n \\ \mathcal{D}_2 \\ B \end{array}}{A}\,/\mathsf{e}$$

$$\rightsquigarrow \quad \begin{array}{c} \mathsf{u}_1 \ \cdots \ \mathsf{u}_n \\ [\mathcal{D}_2/\mathsf{u}_{n+1}]\mathcal{D}_1 \\ A \end{array}$$

and its counterpart for the \-rules, a proof, $\mathcal{D}$, in the Lambek Calculus, is said to be *normal*, if no subderivation of $\mathcal{D}$ matches the left-hand side of either normalization scheme. A normal proof has the property that it uses introduction rules only where necessary.

A proof object for the judgement, "$\mathcal{D}$ is normal," can be constructed in several ways, depending on the last step of the proof of $\mathcal{D}$. First, if the last step of the proof of $\mathcal{D}$ is /e (resp. \e), then we must make sure that the last step of the proof of the left (resp. right) premise is not /i (resp. \i), and that the proofs of both premises are normal (since "normal" is a property that applies to all subderivations):

```
normal: ded L R B -> type.

norm_/e: normal (/e D2 D1 _)
          <- non-/i D1
          <- normal D1
          <- normal D2.
norm_\e: normal (\e D2 D1 _)
          <- non-\i D2
          <- normal D1
          <- normal D2.
```

In the signature, anonymous variables are used for unimportant objects that can be reconstructed by Elf. The types `non-/i` and `non-\i` correspond to judgements that the last step of a deduction is not /i or \i, respectively.

If the last step of the proof of $\mathcal{D}$ is an introduction rule, on the other hand, then the proof of $\mathcal{D}$ itself cannot match either normalisation scheme, but we must still check the subderivations. To do this, we must introduce parameters for the hypothetical node and deduction that its premise requires. Since this hypothetical deduction is also considered a sub-derivation, it must have hypothetical proofs that its last step is not /i, or \i, and that it is a normal subderivation:

```
norm_/i: normal (/i D _)
  <- ({n:nat} {l:less R n} {i:ded R n _}
       non-/i i -> non-\i i -> normal i
       -> normal (D n l i)).
norm_\i: normal (\i D _)
  <- ({n:nat} {l:less n L} {i:ded n L _}
       non-/i i -> non-\i i -> normal i
       -> normal (D n l i)).
```

Notice that, since the deduction of the premise is represented as a higher-order object at the meta-level, we can obtain the subderivation to be proven normal, `(D n l i)`, by beta-reduction at the meta-level. Since LCG derivations are represented by parametric deductions in LF, the theorems proven in the next section will also require such hypothetical proofs for the hypothetical deductions of derivations. In this way, we can assume that both the parameters, and the cases of proofs that apply to them, are provided by the context in which a proof (or parse) object is constructed, rather than carry around explicitly abstracted objects with bound variables.

It is also possible to define a judgement that proves that every deduction is equivalent to a normal deduction. Since the parametric deductions of premises to introduction rules are represented as higher-order objects in the meta-level, this normalization can be achieved simply by beta-reduction at the meta-level, i.e. having matched the left-hand side of a normalization schema, the normalization can be achieved by making the proof substitution, $[\mathcal{D}_2/\mathsf{u}_{n+1}]$, indicated on the right-hand side, using beta-reduction.

The equivalence proof of the next section will also exploit a quality that normal proofs of right-atomic categories have, namely that they only use the `/e` rule and hypothetical deductions. This judgement we call `only_/e`. For the sake of brevity, the signatures for the normal form theorem, `only_/e`, `non-/i` and `non-\i` are not given in this abstract.

6 One Direction of the Weak Equivalence Theorem

Theorem 2 *For any set of strings,* S, *there exists a CFG,* G, *such that* L(G) = S *if and only if there exists an LCG,* L, *such that* L(L) = S.

The proof presented in this paper will prove the "only if" direction only.

The weak equivalence of Ajdukiewicz Categorial Grammar (ACG, basically, LCG without the introduction rules) to CFG's was first proven by Gaifman, and later, more elegantly, by Shamir [2], and even later, even more elegantly by Gladkij [9]. The "only if" direction of Shamir's proof showed how to reduce a Chomsky Normal Form CFG to a kind of ACG, called a Restricted Categorial Grammar (RCG) which restricted the forms of categories it could assign in its initial assignment function.

The "only if" direction of the present theorem was first proven by Cohen [7], by inducing an LCG from an RCG, and demonstrating its equivalence by a series of very unintuitive cases, made all the more unreadable by its reference to a few key lemmata from the context of another proof which was later found to be mistaken. Buszkowski [3], later presented a much cleaner proof by introducing the notion of *order* to categories (based on the depth of occurence of slashes), and proving the weak equivalence of LCG's and ACG's when only first-order categories are used. Since Gladkij's proof transforms a Chomsky Normal Form CFG into an ACG with first-order categories, this also established the result.

The present proof does not utilize Gaifman's theorem at all, but instead constructs an LCG directly from a Greibach Normal Form CFG (i.e. one in which every rule is of the form $N_0 \rightarrow T N_1 \ldots N_q$, for T a terminal, and N_i, non-terminals) which uses only right-atomic categories, a stronger restriction than to first-order categories, which allow \. In fact, this proof can very easily be adapted to prove Gaifman's theorem in a much cleaner fashion than even Gladkij's proof. This proof also uses the natural-deduction formulation of the Lambek Calculus, whereas all previous proofs have used the sequent formulation[3]. Without the benefit of a framework such as the LF formalism, and the implementation of it in which one may prototype such proofs, it was very difficult to conceive of any natural-deduction style proof, let alone one this elegant. The present proof owes both its existence and its correctness to Elf.

Given a CFG, $G = \langle T, N, s, R\rangle$, we define the *induced* Lambek Grammar of G to be $\bar{G} = \langle T, \bar{R}, s\rangle$, where, for every $t \in T$, $\bar{R}(t) = \{N/N_q/\ldots/N_1 \mid N \rightarrow tN_1 \ldots N_q \in R\}$. This LCG operates over a category system generated from the set of basic categories, N, the set of non-terminals.

Now it remains to show that $L(G) = L(\bar{G})$, which follows from two more general theorems.

Theorem 3 *Given a CFG,* G, *symbols* $\mathrm{X}, \mathrm{X}_1, \ldots, \mathrm{X}_\mathrm{n}$, *category* B, *and non-terminals* $\mathrm{N}_1, \ldots, \mathrm{N}_\mathrm{q}$, *if* $\mathrm{X} \Rightarrow^*_\mathrm{G} \mathrm{X}_1 \ldots \mathrm{X}_\mathrm{n}$, *then if* $\mathrm{B} = \bar{\mathrm{X}}$, *an induced cate-*

[3]The present proof can also be conducted in the sequent formulation [20].

gory of X, then there exist $\bar{\mathrm{X}}_1, \ldots, \bar{\mathrm{X}}_\mathrm{n}$ such that:

$$\begin{array}{c} \dfrac{}{\bar{\mathrm{X}}_1}\mathsf{u}_1 \quad \cdots \quad \dfrac{}{\bar{\mathrm{X}}_\mathrm{n}}\mathsf{u}_n \\ \mathcal{D} \\ \bar{\mathrm{X}} \end{array}$$

Proof: The inclusion, $L(G) \subseteq L(\bar{G})$, directly follows from this theorem. In this abstract, I will present only a few cases, along with their implementations. The proof is by induction on the length of the rewriting, k. This judgement must be implemented by a type that is dependent on the CFG derivation and LCG derivation given in the theorem, and also on a third argument, called `symbol_induce` here, that ensures the correct correspondence of objects to actual proof cases: a would-be proof object in which B is not the category corresponding to symbol X in the induced LCG would not correspond to a case of the proof of this theorem, which relates derivations of symbols to proofs of their induced categories only. While this is trivial for non-terminals, which correspond to themselves, a terminal in G corresponds to one of any number of non-atomic categories, as given by the assignment function, $\bar{R}$. Thus, to prove this theorem, it suffices to provide higher-order objects that guarantee, given any first argument `e:edge N M X`, that a well-typed object, `p:e=> e SymInd D` exists. This is implemented as:

```
e=>: edge N M X -> symbol_induce X B
     -> ded N M B -> type.
```

Case 1: $k = 0$ Then $X_1 = X$, so

$$\mathcal{D} = \dfrac{}{\bar{X}}\mathsf{u}_1$$

In the representation of this proof, just as before, parametric deductions are represented as parametric deductions in the meta-language by representing the hypothetical deductions of the categories of words in the given string as variables in the context. As a result, this case of the proof is also represented in the context. A variable for this case of the theorem must occur in a context which contains the variable to which this case applies, plus variables for the other arguments on which it depends. Thus, at the additional layer of representation mentioned above, in which deductions are generalized over the grammar and the given string, we must introduce four variables into the context for each word: a hypothetical deduction of its CFG symbol, `e:edge N (s N) T` (where `s` is the successor function for natural numbers), a hypothetical deduction of its induced LCG category, `d:ded N (s N) B`, a hypothetical deduction that its symbol induces its category, `i:symbol_induce T B`, and a hypothetical deduction of this theorem's proof for this case, `p:e=> e i d`, where `N` is the position of the word in the string (i.e. the number of the node to the edge's left).

Case 2: $k > 0$ So X is a nonterminal and $X \Rightarrow_G T N_1 \ldots N_q \Rightarrow^*_G X_1 \ldots X_n$. There are two subcases here:

Case 2.1: $q = 0$ By the inductive hypothesis, if B is an induced category of T, then its deduction exists; but the induced category of X is an induced category of T since the rule $X \rightarrow T \in R$.

The inductive step can be represented by a recursive call. We also need an object, which will be called `sym_t`, that can create an object of type `symbol_induce X B`, given the context-free rule, `R`, used in the first step of the CFG derivation. `R`, thus, uniquely determines the appropriate type instance of the corresponding LCG derivation.

```
e=>_app_unit: e=> (rule_app_unit ET R _)
                  _ DT
              <- e=> ET (sym_t R) DT.
```

Since the recursive call is on the derivation, `ET`, from a terminal category, it must be satisfied by one of the hypothetical cases from Case 1 that exists in the context. This call thus links information about rule application, which is distributed throughout a CFG rewriting (or parse tree), to information about category assignment in an LCG deduction, which is located entirely in the hypothetical deductions (specifically, it determines which catgory is hypothesized for each word, since a word can be assigned more than one category). This difference in the distribution of phrase structure information is vital to understanding the difference between CFG's and LCG's; and because we represent the induced category of the deduction at the level of the proof (using the intermediate `symbol_induce` argument), this proof representation is operationally capable of transforming a derivation in a CFG to a deduction in an LCG, in spite of this difference. Looking at the difference in this way may also lead to a better understanding of some of the open parsing questions in the Categorial Grammar domain. □

For the opposite inclusion, three lemmata are required, and the fact that every deduction in the Lambek Calculus has a normal form deduction will be used. It should be understood, when I mention a normal parametric deduction, that the `normal` proof for that deduction depends on hypothetical `normal` proofs for its parameters.

Again, I assume that the contexts in which proofs of these lemmata are evaluated have variables corresponding to the hypothetical CFG and LCG derivations for each word in the given string, and also variables for the corresponding cases of the proofs themselves, thus making both the derivations, and the proofs of these lemmata, parametric deductions. These hypothetical cases are, therefore, omitted, since their proofs are simply those variables.

Lemma 4 *In the Lambek Calculus, a normal parametric deduction whose parametric categories are all right-atomic, and whose last step is not* `/i` *or* `\i` *consists only of applications of* `/e`*, and proves a right-atomic category as its conclusion.*

Proof: The proof is by structural induction. This judgement must be represented by a type that depends on the deduction at hand, `D`, one object for each of the three conditions stated, and one object each to prove that both conclusions hold. Operationally, the first four parameters of `lemma4` can be regarded as "input," and the last two, as "output:"

```
lemma4: {D: ded L R B} normal D -> non-\i D
          -> non-/i D -> only_/e D
          -> right_atomic B -> type.
```

Case 2: last step is `/e` The last step of the deduction of the functor (right) category is not `/i` or `\i`; so by the inductive hypothesis, the deduction of the functor category, some A/B, consists only of `/e` applications and A/B is right-atomic.

By the definition of right-atomicity, B is atomic, so the last step of the deduction of the argument (right) category cannot be `/i` or `\i`, since they derive slashed categories. So by the inductive hypothesis, this deduction also consists of only `/e` applications. Since, in this case, the last step is `/e`, we have the first conclusion.

Also by definition of the right-atomicity of A/B, we know that A is right-atomic. This gives us the second conclusion.

Because of the restrictions imposed in the lemma, this is the only (non-hypothesized) case that has a well-typed object, and consists simply of an application of the inductive hypothesis to each branch of the deduction. Again, the inductive steps correspond to recursive calls. The type of the resulting object is assembled from parameters of the types for the inductive steps, using functor objects for the judgements that have been introduced:

```
l4_/e: lemma4 (/e D2 D1 _)
              (norm_/e N2 N1 NF1)  _ _
              (oe OE2 OE1) RightAtomic
        <- lemma4 D1 N1 (n\i_/ Cat/) NF1 OE1
           (strict Atomic2 RightAtomic Cat/)
        <- lemma4 D2 N2 (n\i_at Atomic2)
           (n/i_at Atomic2) OE2
           (atomic Atomic2).
```

`oe` is a functor that constructs `only_/e` objects for a derivation from `only_/e` objects for its substructures. `strict` is a functor that constructs a `right_atomic` proof for a strictly right-atomic category according to the definition. `atomic` constructs a `right_atomic` proof for an atomic category from a proof that the category is indeed atomic. The object, `Cat/`, represents a proof that the category derived by the subdeduction, `D1` (namely, the left premise of the `/e` rule) is a forward-slashed category, which `n\i_/` uses to argue that the last step of `D1` could not have been `\i`. `n\i_at` uses the proof, `Atomic2`, that the resultant category of `D2` is atomic, in the same way. Notice how the "output" right-atomicity proof from the first inductive step contains the right-atomicity proof that we need, and that the "output" only_/e proof is built out of the two only_/e proofs from the inductive steps. □

Lemma 5 *Given symbols,* $\mathrm{X}, \mathrm{X}_1, \ldots, \mathrm{X}_\mathrm{n}$*, and one induced category for each of them, such that* $\bar{\mathrm{X}}_1, \ldots, \bar{\mathrm{X}}_\mathrm{n}$ *are right-atomic, and* $\bar{\mathrm{X}}$ *is atomic, and given non-terminal symbols* $\mathrm{Z}_1, \ldots, \mathrm{Z}_\mathrm{k}$ *for some* $\mathrm{k} \geq 0$*, and a derivation:*

$$\begin{array}{ccc} \overline{\bar{\mathrm{X}}_1}\,\mathsf{u}_1 & \ldots & \overline{\bar{\mathrm{X}}_\mathrm{n}}\,\mathsf{u}_n \\ & \mathcal{D} & \\ & \bar{\mathrm{X}}/\mathrm{Z}_\mathrm{k}/\ldots/\mathrm{Z}_1 & \end{array}$$

such that $\mathcal{D}$ *is normal and consists only of* /e *applications, then* $\mathrm{X} \Rightarrow^*_\mathrm{G} \mathrm{X}_1 \ldots \mathrm{X}_\mathrm{n}\mathrm{Z}_1 \ldots \mathrm{Z}_\mathrm{k}$.

Proof: The proof is by structural induction. Again, to represent this judgement, we need a type that is dependent on the derivations given in the theorem, along with objects for each of the restrictions and an object for the conclusion. Here, however, since we are building a CFG rewriting that begins with X from an LCG deduction that derives a category whose head is X, we need a head_of judgement to ensure the proper correspondence of proofs to objects (just as we used a symbol_induce object in Theorem 3):

```
lemma5: {D:ded L R B} normal D
        -> right_atomic B -> only_/e D
        -> head_of B H NList
        -> edge_tail L R H NList -> type.
```

head_of represents a judgement that a category H is the head of another category B, with arguments in the list of categories, NList. The type edge_tail represents the judgement that non-terminal N rewrites to some string ending in the sequence of non-terminals represented by the list NList. Objects of this type represent the rewritings we obtain as the consequence of this lemma. For the sake of brevity, the constructors of this type and the individual cases of the proof are omitted from this abstract. □

Theorem 6 *Given symbols,* $\mathrm{X}, \mathrm{X}_1, \ldots, \mathrm{X}_\mathrm{n}$*, and one induced category for each of them, such that* $\bar{\mathrm{X}}_1, \ldots, \bar{\mathrm{X}}_\mathrm{n}$ *are right-atomic, and* $\bar{\mathrm{X}}$ *is atomic, and a derivation:*

$$\begin{array}{ccc} \overline{\bar{\mathrm{X}}_1}\,\mathsf{u}_1 & \ldots & \overline{\bar{\mathrm{X}}_\mathrm{n}}\,\mathsf{u}_n \\ & \mathcal{D} & \\ & \bar{\mathrm{X}} & \end{array}$$

such that $\mathcal{D}$ *is normal, then* $\mathrm{X} \Rightarrow^*_\mathrm{G} \mathrm{X}_1 \ldots \mathrm{X}_\mathrm{n}$.

Proof: The inclusion, $L(G) \supseteq L(\bar{G})$ directly follows from this theorem. This is straightforwardly represented by the judgement:

```
e<=: {D: ded L R B} normal D -> atomic B
     -> symbol_induce N B
     -> edge L R (n N) -> type.
```

and straightforwardly proven as follows:

Since $\bar{X}$ is atomic, the last step of $\mathcal{D}$ cannot be /i or \i. So by Lemma 4, $\mathcal{D}$ consists only of /e applications. Thus, by Lemma 5, $X \Rightarrow^*_G X_1 \ldots X_n$.

We can use an object of type lemma4 to obtain an only_/e proof, OE, which can then be used as input to lemma5. Since $\bar{X}$ is atomic, well-typing requires the last parameter of lemma5 to be of type, edge_tail L R N nil, thus yielding the object et_nil applied to the desired rewriting proof, E: □

```
e<=*: e<= D N Atomic _ E
       <- lemma4 D N (n\i_at Atomic)
          (n/i_at Atomic) OE (atomic Atomic)
       <- lemma5 D N (atomic Atomic)
          OE (ho_at Atomic) _ (et_nil E).
```

7 Conclusion

I have improved upon an idea from chart parsing to create an adequate representation for Lambek Categorial Grammar deductions in the LF formalism. Using this, I have found an elegant new proof of one direction of the weak-equivalence theorem for LCG's and CFG's that uses natural deduction, and by-passes the various auxiliary grammars and theorems to which past proofs have had to resort. Using LF, I have found a way to look at CFG and LCG derivations in terms of distribution of rule information, which I believe will yield a cleaner proof of the other direction of this equivalence, and, eventually, an insight into the complexity of parsing with Categorial Grammars. A similar representation can be used to represent sequent-style derivations in LCG, an equivalence proof between the sequent and natural-deduction style calculi for LCG's, and a proof of the cut-elimination theorem for LCG's [20]. LF and Elf have proven to be valuable tools for reasoning about higher-order objects and proofs in this domain; and have proven to be capable of bearing fruit for research not only in Categorial Grammar, but in substructural logics in general, to which the representation given here can easily be adapted. The promise for applications in this domain is even greater for LLF [6], a logical framework and implementation based on linear logic, and demands investigation into creating a sub-linear framework in which inferences would be sensitive to the order of variables in context.

References

[1] Aarts, E., 1995. *Investigations in Logic, Language and Computation.* ILLC Dissertation Series 1995-19, Amsterdam.

[2] Bar-Hillel, Y., Gaifman, C., and Shamir, E., 1960. On Categorial and Phrase Structure Grammars. In *The Bulletin of the Research Council of Israel.*

[3] Buszkowski, W., 1988. Generative Power of Categorial Grammars. In Oehrle et al., eds., *Categorial Grammars and Natural Language Structures.* D. Reidel, Dordrecht.

[4] Carpenter, B., 1995. A Type-Logical Account of Plurality. In *Proceedings of the First International Conference on Formal Grammar*, Barcelona.

[5] Carpenter, B., 1996. *Lectures on Type-Logical Semantics.* MIT Press, Cambridge.

[6] Cervesato, I., and Pfenning, F., 1995. A Linear Logical Framework. Submitted to LICS-96.

[7] Cohen, J.M., 1967. The Equivalence of Two Concepts of Categorial Grammar. *Information and Control.*

[8] Gazdar, G., and Mellish, C.S., 1989. *Natural Language Processing in PROLOG: An Introduction to Computational Linguistics.* Addison-Wesley, Wokingham.

[9] Gladkij, A.V., 1973. *Formal Grammars and Languages* (in Russian). Izdat. "Nauka," Moscow.

[10] Harper, R., Honsell, F., and Plotkin, G., 1993. A Framework for Defining Logics. *Journal of the Association for Computing Machinery.*

[11] Hendriks, H., 1993. *Studied Flexibility: Categories and Types in Syntax and Semantics.* ILLC Dissertation Series 1993-5, Amsterdam.

[12] Hepple, M., 1992. Chart Parsing Lambek Grammars: Modal Extensions and Incrementality. In *Proceedings of COLING-92*, Nantes.

[13] Hodas, J.S. and Miller, D., 1994. Logic Programming in a Fragment of Intuitionistic Linear Logic. *Information and Computation.*

[14] Kanovich, M.I., 1991. The Multiplicative Fragment of Linear Logic is NP-Complete. ITLI Prepublication Series X-91-13, Amsterdam.

[15] Kanovich, M.I., 1991. The Horn Gragment of Linear Logic is NP-Complete. ITLI Prepublication Series X-91-14, Amsterdam.

[16] Koenig, E., 1990. The Complexity of Parsing with Extended Categorial Grammars. In *Proceedings of the 13th International Conference on Computational Linguistics*, Helsinki.

[17] Moortgat, M., 1991. Generalized Quantification and Discontinuous Type Constructors. To appear in Sijtsma and van Horck, eds., *Proceedings of the Tilburg Symposium on Discontinuous Constituency.* De Gruyter, Berlin.

[18] Morrill, G., 1994. *Type-Logical Grammar.* Kluwer, Dordrecht.

[19] Pentus, M., 1992. Lambek Grammars are Context-Free. Technical Report, Dept. of Mathematical Logic, Moscow University.

[20] Penn, G., 1996. An LF Implementation of Gentzen's Theorem for Lambek Categorial Grammar. Technical Report, School for Computer Science, Carnegie Mellon University.

[21] Pereira, F.C.N. and Shieber, S.M., 1987. *Prolog and Natural-Language Analysis.* CSLI Lecture Notes, Stanford.

[22] Pfenning, F., 1991. Logic Programming in the LF Logical Framework. In Huet and Plotkin, eds., *Logical Frameworks.* Cambridge University Press.