LCGbank: A Corpus of Syntactic Analyses Based on Proof Nets

Aditya Bhargava[*], Timothy A. D. Fowler[*†] and Gerald Penn[*]

[*]Department of Computer Science
University of Toronto
{aditya,tfowler,gpenn}@cs.toronto.edu

[†]Cerence Inc.
Toronto, ON

Abstract

In syntactic parsing, *proof nets* are graphical structures that have the advantageous property of invariance to spurious ambiguities. Semantically-equivalent derivations correspond to a single proof net. Recent years have seen fresh interest in statistical syntactic parsing with proof nets, including the development of methods based on neural networks. However, training of statistical parsers requires corpora that provide ground-truth syntactic analyses. Unfortunately, there has been a paucity of corpora in formalisms for which proof nets are applicable, such as Lambek categorial grammar (LCG), a formalism related to combinatory categorial grammar (CCG). To address this, we leverage CCGbank and the relationship between LCG and CCG to develop LCGbank, an English-language corpus of syntactic analyses based on LCG proof nets. In contrast to CCGbank, LCGbank eschews type-changing and uses only categorial rules; the syntactic analyses thus provide fully compositional semantics, exploiting the transparency between syntax and semantics that so characterizes categorial grammars.

Keywords: Lambek categorial grammar, proof net, grammar conversion, English treebank

1. Introduction

In the family of categorial grammars, combinatory categorial grammar (CCG) has received by far the most attention in the computational linguistics literature, including the development of multiple statistical parsers (e.g., Clark and Curran, 2007; Lewis et al., 2016; Stanojević and Steedman, 2020). Another member of the categorial family, Lambek categorial grammar (LCG), has been less well-explored from a statistical perspective. The historical lack of attention is likely due to two notable results: (1) LCG is weakly context-free equivalent (Pentus, 1997); and (2) LCG parsing is NP-complete (Pentus, 2006; Savateev, 2012).

We maintain that neither of these issues is necessarily practically relevant, especially in the context of *statistical* parsing. Context-sensitive parsing is required only for certain specific constructions in a handful of languages; context-free grammars can get quite far, and in practice, statistical parsers for mildly context-sensitive formalisms such as CCG are typically restricted to context-free fragments (Fowler and Penn, 2010). The numerical parameterization of discrete formal grammars, furthermore, is known to have drastic consequences for hierarchies of formal language complexity (Rabin, 1963; Nasu and Honda, 1971), to the extent that even numerical precision can determine the generative capacity of a particular grammar.

As for NP-completeness, time complexity is less relevant for statistical parsers if they can nevertheless return parses quickly. For example, Lewis and Steedman's (2014) parser has worst-case exponential time complexity, and yet is substantially faster than Clark and Curran's (2007) venerable parser even though the latter has cubic time complexity. As well, LCG's time complexity can be shown to be polynomial if a bound is assumed on the order of the lexical categories (Pentus, 2010), which is a reasonable practical assumption (Fowler, 2009a).

The lack of research attention to statistical LCG parsing is especially unfortunate as LCG has a distinct advantage: its derivations can be represented by *proof nets*, graphical structures that compactly encode syntactic analyses such that semantically equivalent (i.e., spuriously ambiguous) derivations are represented by a single proof net. By contrast, there is no known method of applying proof nets to CCG, despite interest in doing so (Buch, 2009). Instead, CCG parsers employ normal-form constraints (Eisner, 1996; Hockenmaier and Bisk, 2010) to avoid spurious ambiguities. With LCG proof nets, invariance to spurious ambiguities is baked in to the representation. For example, the two CCG derivations in Figure 1 correspond to the same LCG proof net, shown in Figure 2.

In at least partial pursuit of proof nets' advantages, recent years have seen renewed interest in statistical syntactic parsing with proof nets, including the development of methods based on neural networks (e.g., Kogkalidis et al., 2020b; Bhargava and Penn, 2021). Of course, training of statistical parsers requires corpora that provide ground-truth syntactic analyses. This is another area in which LCG and related formalisms have been under-

Code to construct LCGbank from CCGbank is available at https://doi.org/10.5683/SP3/ZVGWQY.

LREC-COLING 2024, pages 10225–10236

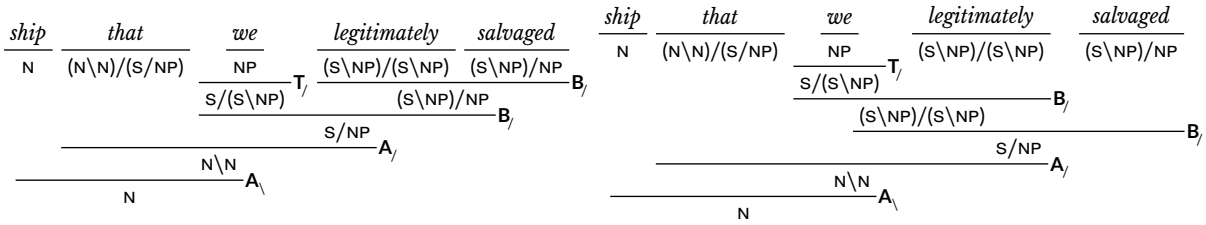

Figure 1: Two CCG derivations of N for the object relative clause *ship that we legitimately salvaged*. Although the derivations are distinct, they are semantically equivalent (i.e., spuriously ambiguous).

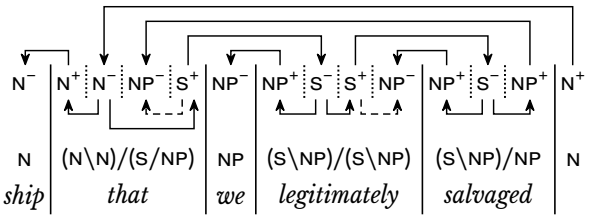

Figure 2: The LCG term graph corresponding to both CCG derivations in Figure 1 *and* the LCG derivation in Figure 4.

$$\frac{\Delta \vdash x/Y \quad \Gamma \vdash Y}{\Delta, \Gamma \vdash x}\ /\mathrm{E} \qquad \frac{\Gamma \vdash Y \quad \Delta \vdash x\backslash Y}{\Gamma, \Delta \vdash x}\ \backslash\mathrm{E}$$

(a) Forward elimination (b) Backward elimination

$$\frac{\Gamma, Y \vdash x \quad (\Gamma \neq \varepsilon)}{\Gamma \vdash x/Y}\ /\mathrm{I} \qquad \frac{Y, \Gamma \vdash x \quad (\Gamma \neq \varepsilon)}{\Gamma \vdash x\backslash Y}\ \backslash\mathrm{I}$$

(c) Forward introduction (d) Backward introduction

$$\frac{}{x \vdash x}\ \mathrm{Ax} \qquad \frac{w \quad ((w, x) \in \text{Lexicon})}{w \vdash x}\ \mathrm{Lex}$$

(e) Axiom (f) Lexicon

Figure 3: The rule schemata of LCG. x and Y are variables over categories while w is a variable over words. Γ and Δ are variables over sequences, where each sequence item is either a category or a word. ε represents the empty sequence.

explored in comparison to CCG: where the last two decades have seen the development of several CCG corpora covering multiple languages and domains, there has been a paucity of corpora to support the development of proof net–based statistical parsers.

In this paper, we leverage the relationship between CCG and LCG to develop a corpus of proof net–based syntactic analyses. Our process begins with CCGbank (Hockenmaier and Steedman, 2005). To get around CCG's incompatibility with proof nets, we first convert CCGbank's derivations to LCG, lexicalizing the incompatible rules. This includes CCGbank's non-categorial type-changing rules, which cloud the otherwise transparent relationship between categorial syntax and semantics. The resultant LCG derivations then directly specify the corresponding proof net.

The result is LCGbank, an English-language corpus that provides purely-categorial syntactic analyses based on proof nets and thus, as a direct consequence, fully compositional semantics. Additionally, LCGbank includes new analyses for the sentences that were excluded from CCGbank's own conversion from the Penn Treebank (PTB).

2. Background

2.1. Lambek categorial grammar

Like CCG, LCG descends from the earlier AB grammar (Ajdukiewicz, 1935; Bar-Hillel, 1953). The set of valid lexical categories is formed by the closure of two binary connectives, the forward slash / and the backward slash \, over a small set of **atomic** categories (or **atoms**) such as S (for sentences) and NP (for noun phrases). Non-atomic categories are called **complex** categories. The connectives create functional categories; the forward slash specifies that an argument must appear to the right while the backward slash specifies that an argument must appear to the left. For example, the category ((S\NP)/NP)/NP combines with two NPs to its right and one NP to its left to yield an S.[1] Thus, in English, this category is used to represent a ditransitive verb.

But where CCG adds additional rule schemata such as composition, LCG treats the slash operators as logical connectives and so defines the basis of a logical system of reasoning over syntactic types, called the Lambek calculus (Lambek, 1958). While there are a number of variants of the Lambek calculus, the one of relevance to our work is known as **L**, the associative Lambek calculus without product and banning empty antecedents.[2] Combining the rules of the calculus with a lexicon yields an LCG, the rule schemata for which are shown in Figure 3.

A statement in L, called a **sequent**, has an ordered list of categories called the **antecedent** on the left of the turnstile (⊢) and a single category

1. Note that our use of the backward slash follows Steedman notation, which is more common in the CCG literature; the same category is instead represented as NP\S/NP/NP in most LCG literature, including the original given by Lambek (1958).

2. We refer the interested reader to (Moot and Retoré, 2012) for formal details of Lambek calculus variants.

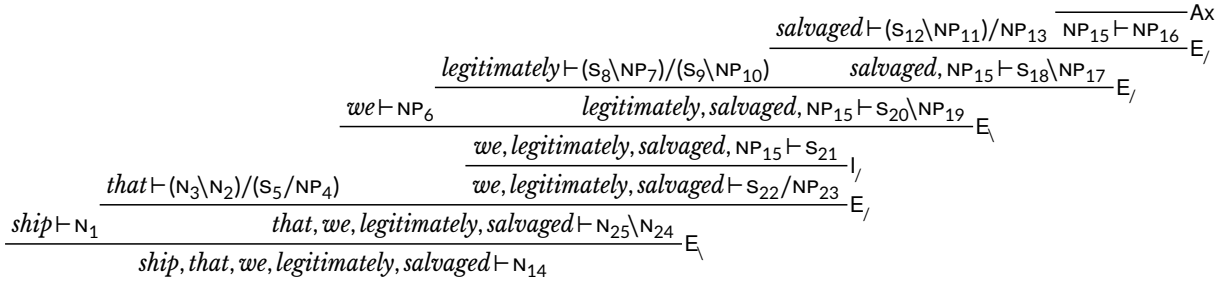

Figure 4: An LCG derivation of N for the object relative clause from Figures 1 and 2. Lexicon rule instances are abbreviated to their conclusions for compactness.

called the **consequent** on the right. The interpretation of a sequent is that its consequent can be derived from its antecedent. The expressions above bars are called **premises** and those below the bars are called **conclusions**. Each rule establishes that its conclusion is true (i.e., derivable) if all of its premises are true. The elimination rules (Figures 3a and 3b) *eliminate* a slashed category, in that it is missing from their conclusions; the introduction rules (Figures 3c and 3d) *introduce* a new slashed category in the consequent of their conclusions. Figure 4 shows an example LCG derivation. (The numerical indices should be ignored for the moment; they will be useful later.)

The elimination rules correspond to the application rules of CCG. Of the other common CCG rules, type-raising and harmonic (i.e., non-crossed) composition, including generalized composition, are derivable in LCG.[3] Thus, CCG derivations that use only these rules are also valid LCG derivations.

On the other hand, substitution and crossed composition rules aren't derivable in LCG. We will discuss this issue further in Section 4.

2.2. Proof nets

Unlike CCG, the rules of the Lambek calculus are a subset of linear logic, where a proof has a graphical representation called a *proof net* (Girard, 1987). Proof nets compactly represent proofs that differ in ways that are considered irrelevant, such as order of rule use where the reading is unaffected. This corresponds directly to the problem of spurious ambiguity: derivations within the same semantic equivalence class are represented by a single proof net; this elegantly and directly encodes the idea that such derivations are equivalent in some way.

There exist different formulations of LCG proof nets. Different proof net versions are equivalent in that they can describe the same logical proofs, but differ in other ways, such as representational overhead. Penn's (2004) presentation of proof nets for L greatly simplified Roorda's (1992) original specification and reduced the number of correctness conditions. Fowler's (2009b) *term graphs* were simpler still, further reducing the number of correctness conditions as well as representational overhead. Term graphs are thus our proof net version of choice; from here, we use the terms "LCG proof net" and "term graph" interchangeably.

3. The harmonic **D** combinator (Hoyt and Baldridge, 2008) is also derivable in LCG, but we are not aware of any CCG corpora that employ it.

3. Term graphs: simplified proof nets

The structure of a term graph can be separated into two components:

1. A *proof frame*, which represents the internal structure of the categories as edges in a directed graph where the category atoms are vertices. A proof frame is constructed by "unfolding" sequent categories categories into linear sequences of atoms. This process is deterministic for a given sequent, and the proof frame is invariant across all proofs of the sequent (if there are any).
2. A *linkage*, which is a set of directed edges between proof frame vertices that connect categorial arguments with their filling categories. A valid LCG derivation deterministically specifies a linkage. Different linkages correspond to semantically distinct derivations (i.e., parses); but different derivations that are spuriously ambiguous (i.e., semantically equivalent) specify the same linkage. For a term graph to be valid, the linkage must satisfy a number of validity conditions (to be described shortly).

The separation between proof frames and linkages is especially relevant for a parser. A non-statistical parser is given a sequent as input, from which the proof frame is constructed deterministically; its job then is to find the set of linkages that yield a valid term graph. On the other hand, a statistical parser is given a sentence without any lexical category assignments. Thus it must first predict a proof frame, which corresponds to the problem of supertagging (Bangalore and Joshi, 1999): predicting lexical categories for the sentence amounts

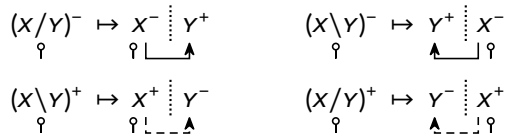

Figure 5: The polarized category decomposition rules. Regular edges are drawn with solid lines while Lambek edges are drawn dashed. The circle-tipped edge shows where any edges (whether regular or Lambek) for the input category are to be attached in the decomposed output categories. The vertical lines specify the order of the outputs.

to predicting a sequent, which then specifies the proof frame. The parser must then find a "best" linkage that yields a valid term graph. This two-stage process directly corresponds to the standard pipeline for statistical CCG parsers, which are usually split into supertagging and derivation phases (e.g., Clark and Curran, 2007; Lewis and Steedman, 2014; Yoshikawa et al., 2017).

3.1. Proof frame construction

The first step in the construction of a proof frame for a sequent is assigning *polarities* to the sequent categories. A category may have either positive or negative polarity: a negatively-charged category x^- indicates that an x is available while a positively-charged category x^+ indicates that an x is sought. So, for a sequent $x_1, x_2, \ldots, x_n \vdash y$, each lexical category x_i of the antecedent is assigned negative polarity (x_i^-) and the consequent category y is assigned positive polarity (y^+).

Next, each polarized category is recursively decomposed according to four rules, shown in Figure 5. These rules "unfold" complex categories into their result and argument subcategories. Subcategorial structure is represented as a directed graph with polarized atoms as vertices and two types of directed edges: **regular** edges and **Lambek** edges.

The end result of this lexical decomposition is a sequence of polarized atoms with directed edges between them as specified by the rules. Note that order is important: the output of each decomposition also specifies a total order over the resultant vertices, which is indicated by the linear, left-to-right placement of the polarized atoms. The resulting graph, including the total order, is the **proof frame** for the sequent $\text{N}, (\text{N}\backslash\text{N})/(\text{S}/\text{NP}), \text{NP}, (\text{S}\backslash\text{NP})/(\text{S}\backslash\text{NP}), (\text{S}\backslash\text{NP})/\text{NP} \vdash \text{N}$ from Figure 4 comprises the polarized atoms and directed edges *below* them in Figure 2.

3.2. Linkages and term graph validity

A **term graph** consists of a proof frame and a valid linkage. A **linkage** is a set of directed edges called **links** that go from positive vertices to negative vertices of the same atomic type (e.g., NP^+ to NP^-). A **regular path** in a term graph is a path consisting only of links and regular edges.

A linkage is **valid** if and only if it forms a perfect matching: each vertex has exactly one link, and that link is outgoing for positive vertices and incoming for negative vertices. In the term graph of Figure 2, the (valid) linkage comprises the directed edges *above* the polarized atoms.

A term graph represents a proof in L, and therefore also an LCG parse, if and only if it meets the following conditions (Fowler, 2009b, 2016):

- T1. The linkage is half-planar; i.e., the links can be drawn above the linearly-ordered vertices without crossing.
- T2. The graph is regular-acyclic; i.e., there is no regular path from a vertex to itself.
- T3. For each Lambek edge $\langle i, j \rangle$, there is a regular path from i to j.
- T4. For each Lambek edge $\langle i, j \rangle$, there is a negative vertex k such that there is a regular path from i to k and there is no Lambek edge $\langle i', k \rangle$ such that there is a regular path from i to i'.

Such a term graph is called a **valid term graph**.

3.3. Converting LCG derivations to term graph linkages

As mentioned above, a derivation for a sequent specifies a linkage over the corresponding proof frame. We translate Roorda's (1992) original specification as required for the simplified representation provided by term graphs. At a high level, the construction of a term graph linkage from an LCG derivation can be separated into three steps:

1. For each LCG rule instance in the derivation, an initial set of vertices and edges (called *match* edges) is constructed according to a set of graph construction rules.
2. Match edges that connect complex categories are recursively propagated until only match edges between atomic vertices remain.
3. Maximal match paths are contracted into links.

3.3.1. Initial graph construction

The rules employed in the first step of term graph construction are shown in Figure 6. Vertices and edges are constructed according to these rules for each LCG rule instance in a derivation. Two details should be noted here. First, unlike the lexical decomposition rules (Figure 5), these graph construction rules do not specify any order over the vertices. Thus the vertices can be placed freely; the particular arrangement shown in Figure 6 is for illustrative purposes only. Second, the edges

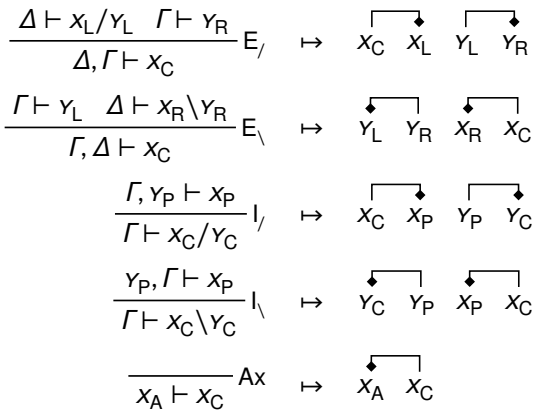

Figure 6: Initial term graph construction rules.

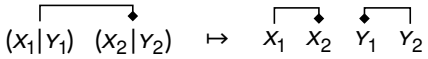

Figure 7: Match edges are propagated through subcategories. | is a variable over {/, \}.

added in this step are not links: they do not (yet) connect positive proof frame vertices to negative ones. We refer to these edges as **match** edges and distinguish them visually by using turned squares to terminate them rather than arrowheads.

As an example, consider again the derivation in Figure 4. Each rule instance in the derivation will produce vertices and edges. To track the vertices, each atom occurring in a consequent is assigned a unique (but otherwise arbitrary) label; we use numerical indices chosen for simple correspondence with the order specified by the proof frame. The same is done for atoms occurring in antecedents of axiom rule instances (NP_{15} in this example), but the labels on an antecedent category are then maintained in subsequent antecedents in the derivation until the category is moved to the consequent via an introduction rule instance. Thus, the initial edges produced from Figure 4 are $NP_{16} \multimap NP_{15}$, $(S_{18}\backslash NP_{17}) \multimap (S_{12}\backslash NP_{11})$, $NP_{13} \multimap NP_{16}$, $(S_{20}\backslash NP_{19}) \multimap (S_8\backslash NP_7)$, $(S_9\backslash NP_{10}) \multimap (S_{18}\backslash NP_{17})$, $S_{21} \multimap S_{20}$, $NP_{19} \multimap NP_6$, $S_{22} \multimap S_{21}$, $NP_{15} \multimap NP_{23}$, $(N_{25}\backslash N_{24}) \multimap (N_3\backslash N_2)$, $(S_5/NP_4) \multimap (S_{22}/NP_{23})$, $N_{14} \multimap N_{25}$, and $N_{24} \multimap N_1$.

3.3.2. Subcategorial match propagation

Next, the match edges connecting complex categories are subcategorially propagated according to the rule in Figure 7. This rule replaces each match edge between complex categories with edges between the categories' subcategories, adding vertices for them if needed. The rule is applied recursively until there remains no match edge between any complex category vertices. Continuing our example, this rule replaces five match edges:

$(S_{18}\backslash NP_{17}) \multimap (S_{12}\backslash NP_{11})$ ↦ $S_{18} \multimap S_{12}$ $NP_{11} \multimap NP_{17}$

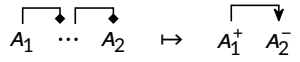

Figure 8: Contraction of maximal match paths to links. A is a variable over *atomic* categories. A_1 must not have any incoming edges and A_2 must not have any outgoing edges for this rule to apply.

$(S_{20}\backslash NP_{19}) \multimap (S_8\backslash NP_7)$ ↦ $S_{20} \multimap S_8$ $NP_7 \multimap NP_{19}$
$(S_9\backslash NP_{10}) \multimap (S_{18}\backslash NP_{17})$ ↦ $S_9 \multimap S_{18}$ $NP_{17} \multimap NP_{10}$
$(N_{25}\backslash N_{24}) \multimap (N_3\backslash N_2)$ ↦ $N_{25} \multimap N_3$ $N_2 \multimap N_{24}$
$(S_5/NP_4) \multimap (S_{22}/NP_{23})$ ↦ $S_5 \multimap S_{22}$ $NP_{23} \multimap NP_4$

3.3.3. Contracting match paths into links

At this stage, the only vertices with exactly one edge (whether incoming or outgoing) correspond to atomic categories from the sequent. In our running example, these are atoms with indices 1 through 9 for the antecedent and index 10 for the consequent. (The antecedent categories are at leaf nodes of the derivation tree, while the consequent category is at the root.) The final step forms links by contracting maximal match paths (Figure 8). In other words, there is a link from vertex *i* to vertex *j* if and only if there is a path of match edges from *i* to *j*. With the linkage so determined, the match edges and intermediate vertices along the contracted paths are discarded. Returning to our example, this procedure yields exactly the linkage shown in Figure 2.

3.4. Related work

Historically, work on syntactic parsing with proof nets has been strictly non-statistical despite the invariance to spurious ambiguities that proof nets provide (Aarts, 1994; Morrill, 1996; Penn, 2004; Fowler, 2010). This is likely due to a lack of corpora providing relevant ground-truth.

This has slowly started to change in recent years, leading to new research on statistical proof net–based parsing and related techniques. For example, Kogkalidis et al. (2020a) developed Æthel, a Dutch-language corpus that provides (among other things) ground-truth proof net parses, extracted automatically from a corpus of dependency parses.[4] Their subsequent parser (Kogkalidis et al., 2020b) relies on Æthel for its training data. As well, recent supertagging research (Margueritte et al., 2023; Kogkalidis and Moortgat, 2023) has made use of TLGbank (Moot, 2015), a French-language corpus of type-logical grammar proofs.

For English, there has been no prior corpus providing proof net–based analyses. The closest formalism to LCG for which an English-language cor-

4. Their formalism is related to LCG but differs notably: their binary connective is non-directional and they incorporate grammatical roles in their categories and rules.

pus exists is CCG, with CCGbank (Hockenmaier and Steedman, 2007) being the largest and most prominent. Directly converting CCGbank's analyses to proof nets is impossible, as there is no known proof net specification for CCG: the crossed rules cause difficulty (Buch, 2009) and substitution is *prima facie* directly incompatible due to the resource-sensitive nature of linear logic. Moreover, CCGbank's non-categorial type-changing rules obscure the transparency between syntax and semantics, making truly compositional semantics difficult to extract from the provided derivations.

4. From CCGbank derivations to LCG

At a high level, the development of LCGbank consists of two steps: (1) alter CCGbank derivations to only use LCG-derivable rules; then (2) convert the resultant derivations into term graphs. The latter step is a straightforward application of the procedures specified above in Section 3; thus, in this section, we focus our discussion on the former.

To convert CCGbank derivations to LCG, those that use non-LCG rules must be re-analyzed. In addition to substitution and any crossed rules, this includes CCGbank's feature-handling and unary type-changing rules. Our overall strategy for this is lexicalization. For binary nodes, for example, when a non-LCG rule is used in a derivation, we alter the node to instead use either forward or backward elimination such that the primary-secondary relationship between the child nodes is preserved.[5] Requisite changes to child node categories are then propagated to the leaf nodes, resulting in changes to the lexical categories assigned to the words in the sentence.

In this section, we discuss representative examples that require such lexicalization. Note that we omit rule names from derivations for compactness.

4.1. Category features

Atomic categories in CCGbank can bear features: S_{dcl} for declarative sentences, S_{to} for to-infinitival clauses, etc. Such features are non-categorial in the sense that the rules for handling them aren't part of any standard categorial grammar; instead, the rules come (in this case) from CCGbank.

For our conversion to LCG, we discard the features entirely and stick to LCG's purely categorial rules. While it may have been possible to maintain at least some of the features through lexicalization, this would be a somewhat unintuitive use of features as there would be no formal relationship between S and S_{dcl}, etc.; effectively, each feature-annotated category would be entirely distinct, with the only relationship between them being the superficial one of having a common initial letter for the category. The potential advantage of maintaining the features is also unclear, as Fowler and Penn (2010) found that removing the features can actually *increase* parsing accuracy.

As they are irrelevant to our conversion, we omit features from CCGbank-based examples here.

5. *Primary* and *secondary* functors are also called *principal* and *subordinate* functors, respectively.

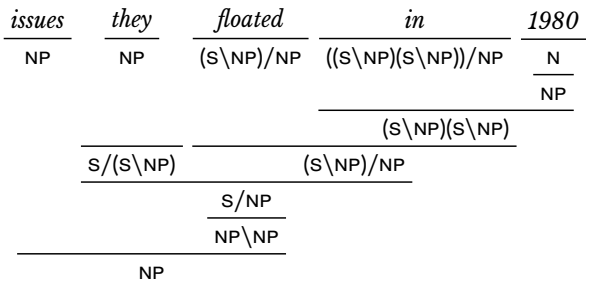

Figure 9: CCGbank's analysis for a contracted fragment of sentence 0339.13, showing its use of type-changing rules NP → N and NP\NP → S/NP.

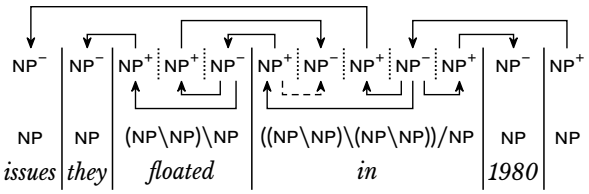

Figure 10: LCGbank's analysis for the reduced relative clause from Figure 9.

4.2. Type-changing rules

CCGbank employs several type-changing rules, allowing some categories to be transmuted as needed. The simplest example of this is the unary NP → N rule. We handle type-changing instances by propagating the higher-level category to the lexical categories, eliminating the type-changing node.

For example, in CCGbank, *related* and *equipment* can be tagged with N/N and N, respectively. Forward application allows N to represent the span *related equipment*, which can then be promoted to NP in order to fill an NP argument slot. Our corresponding LCG derivation propagates the NP to the lexical categories so that *related* and *equipment* are instead tagged with NP/NP and NP, respectively.

CCGbank uses more complex type-changing rules as well. Figure 9 shows how NP\NP → S/NP is used for the reduced relative clauses *the issue they floated in 1980*. Our LCG re-analysis, shown in Figure 10, propagates the NP\NP category so that *floated* has lexical category (NP\NP)\NP and *in* has lexical category ((NP\NP)\(NP\NP))/NP. Where CCGbank employs type-changing to maintain the usual verbal category (S\NP)/NP for *floated*, LCGbank maintains syntax-semantics transparency by

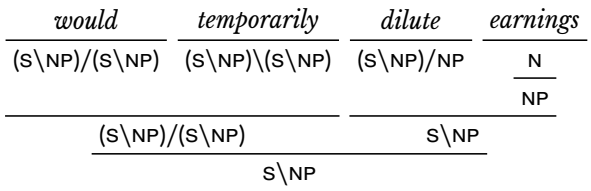

Figure 11: CCGbank's analysis for a fragment of sentence 0317.33, showing its use of crossed composition.

allowing a different verbal lexical category.

4.3. Crossed composition

Of the CCG rules that aren't LCG-derivable, crossed composition is the most frequently used in CCGbank. For example, it is used in CCGbank's analysis of the verb phrase *would temporarily dilute earnings* as shown in Figure 11. This allows *temporarily* to bear the usual lexical category for adverbs ((S\NP)\(S\NP)) while modifying *would*, which has the usual lexical category for auxiliary verbs ((S\NP)/(S\NP)). CCGbank's analysis for the reduced relative clause example (Figure 9) also employs crossed composition, since the (S\NP)\(S\NP) category for *in 1980* must modify the (S\NP)/NP category for *floated* to produce (S\NP)/NP.

While our re-analysis for type-changing fortuitously removes the need for crossed composition in the latter example, this will not be true (or applicable) more generally. Thus, we handle instances of crossed composition by lexicalization: as exemplified in Figure 12, when X\Y composes with Y/Z, we change the former to (Y/Z)\(Y/Z) and propagate the change to the lexical categories.[6] This applies to generalized crossed composition as well.

4.4. Coordination

CCGbank assigns the category conj to coordinators such as *and*. In the grammar, conj is effectively treated as the polymorphic category $(x\backslash x)/x$. In simple cases of *like* coordination, the instantiated coordinating category is evident. For example, the phrase *doors and corners* can be analyzed as having category NP using the nominal coordination rule NP → NP conj NP. It's clear that conj functions as (NP\NP)/NP, since its result and two arguments all have category NP.

Unlike coordination, where the two conjuncts have differing syntactic types, is handled less simply in CCGbank. In the verb phrase *will come a little richer and in a larger amount*, the left conjunct *a little richer* is analyzed as having category S\NP while the right conjunct has category PP. CCGbank adds extra rules in order to maintain the conj category for the coordinator *and*. In this example, the relevant rule is S\NP → S\NP conj PP.

In LCGbank, we do not use polymorphic categories as doing so would require extra (non-LCG) rules for how to handle them.[7] Instead, the category assigned to the coordinator is deduced from the categories of the conjuncts and that of the parent node. For like coordination, this results in a category that matches the $(x\backslash x)/x$ template, such as (NP\NP)/NP for nominal coordination. For unlike coordination, the resulting category does not match the template: for the example above, the conj category is replaced with ((S\NP)\(S\NP))/PP.

4.5. Manual annotations and new analyses

Our conversion process is semi-automated: we developed a framework to categorize the rules used in CCGbank derivations and then altered them with rules such as those described above. This leaves approximately 500 rules without valid LCG derivations; we annotate these manually. Most of these we judged to be annotation errors in CCGbank.

During CCGbank's own conversion from the PTB, 274 sentences were left out because they were problematic for the conversion process (Hockenmaier and Steedman, 2007). For example, cases of sentential gapping were excluded entirely; by contrast, we are able to handle these cases in LCGbank since, as described above, we do not insist that coordinators must always have categories that match the standard $(x\backslash x)/x$ template. LCGbank thus includes new analyses for the 274 excluded sentences. We generate initial analyses using Petrov and Klein's (2007) parser after training it on the converted derivations for the non-excluded sentences. We then manually verify and correct its outputs.

Lastly, we found 40 sentences where the ground-truth term graph had intra-lexical links (i.e., links between vertices that are part of the same lexical category). These were all cases of argument cluster coordination (e.g., *This has both made investors uneasy and the corporations more vulnerable*), where the conjuncts are type-raised categories that must be handled by the coordinator. As Bhargava and Penn (2021) found that disallowing intra-lexical links is a useful search space restriction, we reanalyzed these sentences similarly to how we handled the cases of complex coordination mentioned above, thereby removing the intra-lexical links. For these 40 sentences, we keep the original term

6. It is nearly always the case that the child nodes of the crossed composition rule node are already leaf nodes.

7. A dedicated (ternary) coordination rule is often included in CCG. CCGbank allows only unary and binary nodes, so it provides a simple binarized form of instances of this rule. We ignore this binarization in this paper for simplicity.

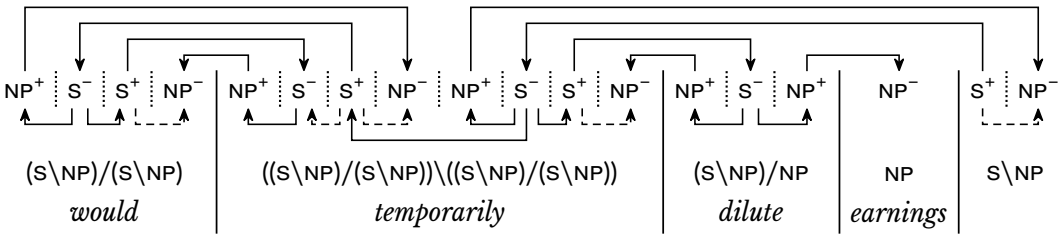

Figure 12: LCGbank's analysis for the verb phrase from Figure 11.

graph (*with* intra-lexical links) in the corpus and provide our re-analysis as an extra alternative.

5. LCGbank: a corpus of LCG proof net parses

The result of our procedures described above is LCGbank, an English-language corpus of proof net–based syntactic analyses for 49,206 sentences. For each sentence, the corpus includes:

1. Lexical category assignments for each word token in the sentence.[8] These lexical categories form the antecedent of the sequent for the sentence.
2. The sentential (i.e., spanning) category for the sentence (usually S).[9] This category is the consequent of the sequent for the sentence.
3. The linkage for the sentence's sequent.

We do not specify the proof frame since it is constructed deterministically from the sequent and thus can be easily determined from the lexical category assignments. Similarly, we do not provide the compositional semantics indicated by the term graphs; the structure of the graph itself represents this and can be converted automatically to other forms (e.g., λ-calculus expressions) as desired by the user.

We follow the CCGbank (and PTB) tradition and designate section 0 as the development/validation set and section 23 as the test set. For the training set, however, we include all of the remaining data (sections 1–22 and 24), in contrast to the usual training set for CCGbank (sections 2–21). This is simply to enable full use of all available data.

As CCGbank is available only via a paid licence, we are unable to provide LCGbank on its own, since it is a CCGbank derivative. Our solution is to instead provide a set of scripts that, when given a copy of CCGbank, produces the files that constitute the LCGbank corpus. Our scripts are freely available under the Apache License, version 2.0 at the URL provided on the first page of this paper.

	C	Cf	L
Sentences	44,614	44,614	44,870
Lexical categories	1,327	487	1,071
Atomic categories	34*	11*	5
Avg. cat order	1.748	1.916	2.317
Exp. cat order	0.785	0.785	0.919
Word types	46,757	44,210	46,946
Uncased	41,599	41,599	41,769
Word-cat pairs	79,527	69,705	91,416
Uncased	73,523	68,165	85,323
Avg. cats per word	1.701	1.577	1.947
Uncased	1.767	1.639	2.043
Exp. cats per word	20.083	14.958	29.731
Uncased	21.889	16.916	31.599

*Six of the atomic categories are for punctuation.

Table 1: Basic statistics for sections 1–22 and 24 of CCGbank (C), CCGbank without features (Cf), and LCGbank (L). Expected values are frequency-weighted while average values are not.

5.1. Corpus statistics

Table 1 shows some basic statistics for CCGbank, CCGbank without features, and LCGbank, computed using LCGbank's training split.[10] At first blush, it would seem that LCGbank has fewer unique lexical categories than CCGbank. But this is mostly due to the removal of features; when comparing against CCGbank *without* features, it is clear that our conversion process more than doubled the number of unique lexical categories.

This is as expected given that our strategy was to lexicalize out the incompatible rules. The new categories accordingly increase lexical ambiguity (categories per word). As we can see when comparing Figures 11 and 12, new categories sometimes take other categories whole as arguments and thus

8. In CCGbank, punctuation that does very little grammatical work (for example, commas that delineate quotes or colons that separate object clauses from their verbs) is treated as a generic adjunct, effectively being "absorbed" into a neighbouring category. For LCGbank, we omit lexical categories for such "transparent" punctuation.

9. In order of decreasing frequency, the sentential categories that appear in LCGbank are S, NP, S\NP, and PP. This is as in CCGbank (minus features), except for five cases which had S_{dcl}/NP; we judged these to be misannotations and re-analyzed them for LCGbank.

10. See Table 2 in Appendix A for the same statistics computed using CCGbank's training split.

are likely to be higher-order than the ones they replace, leading to the notably higher average and expected category order for LCGbank as compared to CCGbank.

As well, disallowing the polymorphic $(x\backslash x)/x$ category drastically increased the number of lexical category pairings for coordinators. In CCGbank sections 1–22 and 24, *and* is assigned category conj for 17,806 of its 18,467 occurrences, with some 33 categories making up the remainder. But for LCGbank, *and* is the single word type with the greatest lexical ambiguity, having 177 different lexical categories assigned to it throughout the training set. Similarly, *or* went from having conj in 2,789 of its 2,922 occurrences and 16 categories making up the remainder to having 68 different lexical categories assigned to it throughout the training set.[11]

As we saw in Section 4, maintaining usual category assignments is a motivating factor behind many of the non-categorial rules in CCGbank. Lexicon size was an explicit consideration at the time that CCGbank was developed: the supertagger's job is to restrict the search space, and a larger lexicon makes predicting lexical category assignments more difficult. We're not concerned by LCGbank's larger lexicon, however, as modern supertaggers are substantially more capable than what was available 20 years ago (e.g., Kogkalidis and Moortgat, 2023; Yamaki et al., 2023).

The increase in lexical ambiguity brings with it the danger of entirely new word-category pairings (relative to what occurs in the training data). In LCGbank's development set, 3.0% of word tokens have categories assigned to them that they never have in sections 2–21. This is a substantial increase from CCGbank, where the number is 2.2% (1.8% if features are discarded). Some of these new pairings even have categories that do not appear at all in the training data. Here, as well, we do not see this as a major downside, as many modern supertaggers are constructive (e.g., Bhargava and Penn, 2020; Prange et al., 2021; Kogkalidis and Moortgat, 2023), building lexical categories from primitives rather than predicting them as wholes.

6. Conclusion

In this paper, we presented LCGbank, an English-language corpus of proof net–based syntactic parses. Our conversion from CCGbank aimed to preserve the transparency between syntax and semantics provided by categorial grammars; LCGbank thus eschews type-changing and its derivations yield fully compositional semantics. We hope that LCGbank proves useful for the development of statistical parsers as well as other research areas related to syntax and proof nets.

6.1. Future work

6.1.1. Dependencies

While our conversion to LCG was based on derivations, CCGbank also provides ground-truth dependencies for the sentences in the corpus. In fact, testing dependencies is the standard method for evaluating statistical parsers trained on CCGbank. These dependencies are meant to represent the predicate-argument structure of the sentence.

One of the major advantages of the dependency-based evaluation is its invariance to spurious ambiguities; but of course, proof nets elegantly handle this issue in LCG's case, and the standard dependency evaluation for CCGbank is not without its own shortcomings (Bhargava and Penn, 2023). Still, accurately predicting predicate-argument relations is useful, and cannot be determined from a derivation alone—statistical CCG parsers incorporate a separate set of rules to convert derivations to dependencies, usually those of Clark and Curran (2007) or Lewis and Steedman (2014). Porting CCGbank's dependencies to LCGbank may warrant further investigation, as the graphical structure of proof nets may facilitate the accurate prediction of dependencies. Moreover, as there is (yet) no standard metric for evaluating statistical LCG parsers, dependencies may prove useful in this regard, and may also enable inter-formalism comparisons, especially between CCG and LCG.

6.1.2. CCGrebank

CCGrebank (Honnibal et al., 2010) provides a number of improvements over CCGbank. For example, corrections are made to how punctuation is handled, a new PT atomic category is added for use with verb-particle constructions, adnominals are made restrictive, etc. Unfortunately, the corpus is not readily available. This (in part) motivated our choice of basing LCGbank on the standard CCGbank, as we can expect far more researchers to have the latter to use as a base for conversion. Still, CCGrebank's improved analyses would be preferable to use, as any conversion would benefit from its corrections. We thus hope to adapt our conversion method to work with CCGrebank as well.

7. Bibliographical references

Erik Aarts. 1994. Proving theorems of the second order Lambek calculus in polynomial time. *Studia Logica*, 53(3):373–387.

Kazimierz Ajdukiewicz. 1935. Die syntaktische Konnexität. In Storrs McCall, editor, *Polish logic,*

11. The most common category for both *and* and *or* in the LCGbank training set is (NP\NP)/NP.

1920–1939, pages 207–231. Clarendon Press, Oxford, United Kingdom. Translated from *Studia Philosophica*, 1:1–27.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

Yehoshua Bar-Hillel. 1953. A quasi-arithmetical notation for syntactic description. *Language*, 29(1):47–58.

Aditya Bhargava and Gerald Penn. 2020. Supertagging with CCG primitives. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 194–204, Online. Association for Computational Linguistics.

Aditya Bhargava and Gerald Penn. 2021. Proof net structure for neural Lambek categorial parsing. In *Proceedings of the 17th International Conference on Parsing Technologies and the IWPT 2021 Shared Task on Parsing into Enhanced Universal Dependencies (IWPT 2021)*, pages 13–25, Online. Association for Computational Linguistics.

Aditya Bhargava and Gerald Penn. 2023. Decomposed scoring of CCG dependencies. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1030–1040, Toronto, Canada. Association for Computational Linguistics.

Armin Buch. 2009. Mildly non-planar proof nets for CCG. In *European Summer School for Logic, Language, and Information Student Session*, pages 160–169, Bordeaux, France.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

Jason Eisner. 1996. Efficient normal-form parsing for combinatory categorial grammar. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 79–86, Santa Cruz, California, USA. Association for Computational Linguistics.

Timothy A. D. Fowler. 2009a. Parsing CCGbank with the Lambek calculus. In *Workshop on Parsing with Categorial Grammars*, pages 38–42, Bordeaux, France.

Timothy A. D. Fowler. 2009b. Term graphs and the NP-completeness of the product-free Lambek calculus. In *Formal Grammar*, pages 150–166, Berlin, Heidelberg. Springer Berlin Heidelberg.

Timothy A. D. Fowler. 2010. A polynomial time algorithm for parsing with the bounded order Lambek calculus. In *The Mathematics of Language*, Lecture Notes in Computer Science, pages 36–43, Berlin, Germany. Springer.

Timothy A. D. Fowler. 2016. *Lambek categorial grammars for practical parsing*. Ph.D. thesis, University of Toronto, Toronto, Canada.

Timothy A. D. Fowler and Gerald Penn. 2010. Accurate context-free parsing with combinatory categorial grammar. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 335–344, Uppsala, Sweden. Association for Computational Linguistics.

Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science*, 50(1):1–101.

Julia Hockenmaier and Yonatan Bisk. 2010. Normal-form parsing for combinatory categorial grammars with generalized composition and type-raising. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 465–473, Beijing, China. Coling 2010 Organizing Committee.

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

Matthew Honnibal, James R. Curran, and Johan Bos. 2010. Rebanking CCGbank for improved NP interpretation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 207–215, Uppsala, Sweden. Association for Computational Linguistics.

Frederick Hoyt and Jason Baldridge. 2008. A logical basis for the D combinator and normal form in CCG. In *Proceedings of ACL-08: HLT*, pages 326–334, Columbus, Ohio. Association for Computational Linguistics.

Konstantinos Kogkalidis and Michael Moortgat. 2023. Geometry-aware supertagging with heterogeneous dynamic convolutions. In *Proceedings of the 2023 CLASP Conference on Learning with Small Data (LSD)*, pages 107–119, Gothenburg, Sweden. Association for Computational Linguistics.

Konstantinos Kogkalidis, Michael Moortgat, and Richard Moot. 2020a. ÆTHEL: Automatically extracted typelogical derivations for dutch. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 5257–5266, Marseille, France. European Language Resources Association.

Konstantinos Kogkalidis, Michael Moortgat, and Richard Moot. 2020b. Neural proof nets. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 26–40, Online. Association for Computational Linguistics.

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

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. LSTM CCG parsing. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 221–231, San Diego, California. Association for Computational Linguistics.

Mike Lewis and Mark Steedman. 2014. A* CCG parsing with a supertag-factored model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 990–1000, Doha, Qatar. Association for Computational Linguistics.

Gaëtan Margueritte, Daisuke Bekki, and Koji Mineshima. 2023. Multi-purpose neural network for French categorial grammars. In *Proceedings of the 15th International Conference on Computational Semantics*, pages 78–82, Nancy, France. Association for Computational Linguistics.

Richard Moot. 2015. A type-logical treebank for French. *Journal of Language Modelling*, 3(1):229–264.

Richard Moot and Christian Retoré. 2012. *The Logic of categorial grammars: A deductive account of natural language syntax and semantics*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg.

Glyn Morrill. 1996. Memoisation of categorial proof nets: parallelism in categorial processing. In *Proofs and Linguistic Categories: Proceedings 1996 Roma Workshop*, pages 157–169.

Masakazu Nasu and Namio Honda. 1971. A context-free language which is not acceptable by a probabilistic automaton. *Information and Control*, 18(3):233–236.

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

Mati Pentus. 1997. Product-free Lambek calculus and context-free grammars. *The Journal of Symbolic Logic*, 62(2):648–660.

Mati Pentus. 2006. Lambek calculus is NP-complete. *Theoretical Computer Science*, 357(1-3):186–201.

Mati Pentus. 2010. A polynomial-time algorithm for Lambek grammars of bounded order. *Linguistic Analysis*, 36(1-4):441–472.

Slav Petrov and Dan Klein. 2007. Improved Inference for Unlexicalized Parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 404–411, Rochester, New York. Association for Computational Linguistics.

Jakob Prange, Nathan Schneider, and Vivek Srikumar. 2021. Supertagging the long tail with tree-structured decoding of complex categories. *Transactions of the Association for Computational Linguistics*, 9:243–260.

Michael O. Rabin. 1963. Probabilistic automata. *Information and Control*, 6(3):230–245.

Dirk Roorda. 1992. Proof nets for Lambek calculus. *Journal of Logic and Computation*, 2(2):211–231.

Yury Savateev. 2012. Product-free Lambek calculus is NP-complete. *Annals of Pure and Applied Logic*, 163(7):775–788.

Miloš Stanojević and Mark Steedman. 2020. Max-margin incremental CCG parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4111–4122, Online. Association for Computational Linguistics.

Ryosuke Yamaki, Tadahiro Taniguchi, and Daichi Mochihashi. 2023. Holographic CCG parsing. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 262–276, Toronto, Canada. Association for Computational Linguistics.

Masashi Yoshikawa, Hiroshi Noji, and Yuji Matsumoto. 2017. A* CCG parsing with a supertag and dependency factored model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 277–287, Vancouver, Canada. Association for Computational Linguistics.

8. Language resource references

Aditya Bhargava, Timothy A. D. Fowler, and Gerald Penn. 2024. *LCGbank*. University of Toronto Dataverse, ISLRN 454-336-063-423-0. PID https://doi.org/10.5683/SP3/ZVGWQY.

	C	Cf	L
Sentences	39,604	39,604	39,831
Lexical categories	1,285	470	1,020
Atomic categories	34*	11*	5
Avg. cat order	1.739	1.902	2.300
Exp. cat order	0.785	0.785	0.919
Word types	44,210	44,210	44,383
Uncased	39,384	39,384	39,541
Word-cat pairs	74,667	69,705	85,686
Uncased	69,114	64,151	80,066
Avg. cats per word	1.689	1.577	1.931
Uncased	1.755	1.629	2.025
Exp. cats per word	19.185	14.958	28.226
Uncased	20.855	16.231	29.987

*Six of the atomic categories are for punctuation.

Table 2: Basic statistics for sections 2–21 of CCGbank (C), CCGbank without features (Cf), and LCGbank (L). Expected values are frequency-weighted while average values are not.

Julia Hockenmaier and Mark Steedman. 2005. *CCGbank*. Linguistic Data Consortium, ISLRN 181-921-208-336-7. PID https://doi.org/10.35111/a589-6d76.

A. Statistics on CCGbank's training split

As discussed in Section 5, LCGbank's training set additionally includes all sections from CCGbank that were unassigned to any split. The statistics in Table 1 were computed over LCGbank's using training set split; to facilitate comparisons against CCGbank's usual training set, Table 2 provides the same statistics, but computed over the standard CCGbank training split (i.e., sections 2–21).