Providing Robustness for a CCG System

Julia Hockenmaier Gann Bierner Jason Baldridge

University of Edinburgh
Division of Informatics
{julia,gbierner,jmb}@cogsci.ed.ac.uk

Abstract

We demonstrate ways to preserve the advantages of using a symbolic grammar formalism as the basis of an NLP system while enhancing its robustness. We automatically acquire a CCG lexicon, combine it with semantic and morphological information from another hand-built, underspecified lexicon, and integrate it with statistical preprocessing methods.

1 Introduction

In this paper, we explore ways of augmenting an existing NLP system which uses Combinatory Categorial Grammar (CCG) (Steedman, 2000) with data-intensive methods and machine learning techniques in order to make it more robust. Specifically, we automatically acquire a CCG lexicon, combine it with semantic and morphological information from another hand-built, underspecified lexicon, and integrate it with statistical preprocessing methods. The techniques we discuss are useful to other symbolic formalisms such as Tree-Adjoining Grammar (TAG) (Joshi, 1988), Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994), and Lexical-Functional Grammar (LFG) (Kaplan and Bresnan, 1982).

CCG and TAG are mildly context-sensitive formalisms (Vijay-Shanker and Weir, 1994) which have nice computational properties such as full lexicalization and worst-case polynomial time parsing (Vijay-Shanker and Weir, 1994). Being fully lexicalized, they have an essentially invariant rule component, which reduces the task of grammar development to that of creating a lexicon. CCG has further benefits, such as semantic transparency and an elegant treatment of coordination. Also, CCG encodes subcategorization information and syntactic potential in a compact and simple representation. Finally, as Doran and Srinivas (to appear) point out, CCG lexicons are generally more compact than TAG lexicons.

Efforts have been made to create wide-coverage lexicons for both TAG and CCG. For example, the XTAG project (XTAG-group, 1999) has built a substantial English TAG lexicon which covers an extensive number of English constructions. In addition, Xia (1999) has acquired TAG lexicons from the University of Pennsylvania Treebank (Marcus *et al.*, 1993). Doran and Srinivas (to appear) induced a CCG lexicon from the XTAG English grammar by mapping TAG trees to CCG categories. Villavicencio (1997) did a semi-automatic translation of the Alvey Natural Language Tools English grammar (Grover *et al.*, 1993) to create a large CCG lexicon.

All of these grammars have been called *wide-coverage*, but their use of the term wide-coverage seems to pertain more to the ability to handle a large variety of linguistic constructions rather than the ability to parse significant portions of large corpora. The latter notion of wide-coverage corresponds much more with the view generally held among statistical NLP researchers. For them, one of the foremost concerns of parsing should be the ability to assign a linguistic analysis (structure, derivation, etc.) to virtually all the data one processes, giving less importance to issues such as grammaticality. Indeed, these are the sort of capabilities which are needed for handling a great deal of the text which exists in the world. In this sense, linguistically motivated systems such as XTAG are quite fragile and require rather controlled input. In order to overcome some of these shortcomings, researchers have turned their attention to the use of corpora in order to extract lexical and syntactic information. Such techniques have been used to create grammars automatically for TAG (Xia, 1999) and semi-automatically for LFG (Kuhn *et al.*, 1998; van Genabith *et al.*, 1999).

Our goal is to build a system which does not suffer from the fragility of many hand-built systems and which benefits from the use of a principled grammar formalism like CCG. Our current system is constructed with components from the Grok library, which was initially a small CCG system for implementing syntactic and semantic analyses. We have taken several measures towards making the library a practical basis for building robust NLP systems. By robust, we mean the ability to parse real world text with significant speed, accuracy, and coverage and produce semantic representations with those parses.

We begin with a brief introduction to CCG. Next, we give a description of an underspecified lexicon which contains hand-coded linguistic specifications for closed lexical classes and basic category and semantic frame information for open ones. Then in §4 and §5 we describe how we acquire a much larger syntactic lexicon and then endow it with the morphological and semantic information from the original lexicon. In §6, we discuss how we can make the system more robust through well known, statistical preprocessing techniques. We end with conclusions and future work.

2 Introduction to CCG

CCG was introduced by Ades and Steedman (1982) as a generalization of the Categorial Grammars (CG) of Ajdukiewicz (1935) and Bar-Hillel (1953). CCG grammars consist of two main parts: a set of combinatory rules and a lexicon of words and their associated syntactic types and semantic interpretations. Examples of lexical entries are given in (1).

(1) a. **John** $\vdash$ NP : $\texttt{john}$

b. **spam** $\vdash$ NP : $\texttt{spam}$

c. **likes** $\vdash$ (S\NP)/NP : $\lambda x.\lambda y.\texttt{like}(y, x)$[1]

The lexical item appears to the left of the $\vdash$ operator, and the right side specifies the syntactic type and the semantic interpretation separated by the colon operator. The backward \ and forward / slashes in the syntactic category of (1c) indicate the directions (left and right, respectively) in which the category for the verb **likes** must find its NP arguments.

During derivations, categories are retrieved from the lexicon and put together by the combinatory rules of CCG. CCG shares two rules with basic CG, those of function application, given in (2).

(2) a. *Forward Application* (>):

$\mathsf{X/Y} : f \quad \mathsf{Y} : a \quad \Rightarrow \quad \mathsf{X} : fa$

[1] See Steedman (1996) for a discussion of the use of lambda notation for CCG interpretations.

b. *Backward Application* ($<$):
$\mathsf{Y} : a \quad \mathsf{X}\backslash\mathsf{Y} : f \quad \Rightarrow \quad \mathsf{X} : fa$

The juxtaposition fa indicates the application of the semantic function f to the argument a. With these rules and with the lexical entries given in (1), the following derivation may proceed, yielding the category S (a sentence) from the string *John likes spam*, with the appropriate interpretation.

(3)
$$\dfrac{\dfrac{\textbf{John}}{\mathsf{NP} : \mathtt{john}} \quad \dfrac{\dfrac{\textbf{likes}}{(\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP} : \lambda x.\lambda y.\mathtt{like}(y,x)} \quad \dfrac{\textbf{spam}}{\mathsf{NP} : \mathtt{spam}}}{\mathsf{S}\backslash\mathsf{NP} : \lambda y.\mathtt{like}(y,\mathtt{spam})}>}{\mathsf{S} : \mathtt{like}(\mathtt{john},\mathtt{spam})}<$$

CCG adds to the basic CG base by adding three classes of rules, each of which corresponds to one of the simplest combinators of Curry and Feys (1958). The three combinators incorporated into CCG are composition, type-raising, and substitution, abbreviated as **B**, **T**, and **S** respectively. Only composition and type-raising are relevant for present purposes.

(4) Rules corresponding to the composition combinator **B**.

a. *Forward Composition* ($>$**B**):
$\mathsf{X}/\mathsf{Y} : f \quad \mathsf{Y}/\mathsf{Z} : g \quad \Rightarrow_{\mathbf{B}} \quad \mathsf{X}/\mathsf{Z} : \lambda x.f(gx)$

b. *Forward Crossing Composition* ($>\mathbf{B}_\times$):
$\mathsf{X}/\mathsf{Y} : f \quad \mathsf{Y}\backslash\mathsf{Z} : g \quad \Rightarrow_{\mathbf{B}} \quad \mathsf{X}\backslash\mathsf{Z} : \lambda x.f(gx)$

c. *Backward Composition* ($<$**B**):
$\mathsf{Y}\backslash\mathsf{Z} : g \quad \mathsf{X}\backslash\mathsf{Y} : f \quad \Rightarrow_{\mathbf{B}} \quad \mathsf{X}\backslash\mathsf{Z} : \lambda x.f(gx)$

d. *Backward Crossing Composition* ($<\mathbf{B}_\times$):
$\mathsf{Y}/\mathsf{Z} : g \quad \mathsf{X}\backslash\mathsf{Y} : f \quad \Rightarrow_{\mathbf{B}} \quad \mathsf{X}/\mathsf{Z} : \lambda x.f(gx)$

(5) Rules corresponding to the type-raising combinator **T**.[2]

a. *Forward Type-raising* ($>$**T**):
$\mathsf{X} : a \quad \Rightarrow_{\mathbf{T}} \quad \mathrm{T}/(\mathrm{T}\backslash\mathsf{X}) : \lambda p.pa$

b. *Backward Type-raising* ($<$**T**):
$\mathsf{X} : a \quad \Rightarrow_{\mathbf{T}} \quad \mathrm{T}\backslash(\mathrm{T}/\mathsf{X}) : \lambda p.pa$

These rules can be restricted to given types for a given language. They bring CCG's weak generative capacity from the context-free level of basic CG to mild context-sensitivity (Joshi *et al.* 1991). This allows a wider range of syntactic analyses, including the famous Dutch crossing dependencies (Steedman, 1985, 2000). They also allow derivations to proceed in many semantically equivalent ways, as the alternative to derivation (3), given below, demonstrates.

(6)
$$\dfrac{\dfrac{\dfrac{\dfrac{\textbf{John}}{\mathsf{NP} : \mathtt{john}}}{\mathrm{T}/(\mathrm{T}\backslash\mathsf{NP}) : \lambda p.p\ \mathtt{john}}>\mathbf{T} \quad \dfrac{\textbf{likes}}{(\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP} : \lambda x.\lambda y.\mathtt{like}(y,x)}}{\mathsf{S}/\mathsf{NP} : \lambda x.\mathtt{like}(\mathtt{john},x)}>\mathbf{B} \quad \dfrac{\textbf{spam}}{\mathsf{NP} : \mathtt{spam}}}{\mathsf{S} : \mathtt{like}(\mathtt{john},\mathtt{spam})}>$$

[2] T in these rules is a variable over categories.

We thus derive the same result as before with a different derivational history. It should be noted that, unlike many formalisms, syntactic derivation is not a representational level in CCG. Rather, it is only a record of the steps which were taken in combining the categories which were introduced into the derivation. The only true level of representation is Logical Form, upon which binding relations are defined.

Extensive introductions and discussions of the linguistic properties and motivations of the CCG formalism can be found in Steedman (1987, 1996, 2000).

3 An Underspecified Lexicon

We begin by describing an underspecified, hand-built lexicon as a starting point for our system. This lexicon consists of entries for closed class words such as prepositions and determiners. In addition, categories are included for open class words, but the words themselves are not listed in the lexicon and are instead supplied by a morphological analyzer. Given a lexical item, the morphological analyzer provides one or more possibilities for the item's part of speech, stem, and morphological features. These features are then percolated into the lexicon for every entry of the given stem and part of speech. Thus, a simple lexical entry such as (7a) specifies the syntactic and semantic category for a stem. This, together with the information provided by the morphological analyzer (7b), expands to the categories in (7c).

(7) a. **walk** $\vdash \mathsf{S\backslash NP} : \lambda x.\mathtt{walk}(x)$

b.

walks:	V	walk	3sg pres
walked:	V	walk	past
	V	walk	pparticiple

c. **walks** $\vdash \mathsf{S\{num = s, per = 3, tense = pres\}\backslash NP} : \lambda x.\mathtt{walk}(x)$
walked $\vdash \mathsf{S\{tense = past\}\backslash NP} : \lambda x.\mathtt{walk}(x)$
walked $\vdash \mathsf{S\{tense = ppart\}\backslash NP} : \lambda x.\mathtt{walk}(x)$

However, as noted above, the lexicon is underspecified, and thus we do not have specific lexical entries like (7a). Instead, we use the notion of *families* from XTAG (XTAG-group, 1999) to organize the hand-built lexicon into collections of categories. Each family is marked with an associated part of speech corresponding to the POS tags produced by the morphological analyzer. Thus, rather than lexical entries for each open class word, we have open class families such as (8). The semantics for these entries are patterns which leave the predicate, P, unspecified.

(8)

Intransitive Family:	V	$\mathsf{S\backslash NP}$	$\lambda x.P(x)$
Transitive Family:	V	$\mathsf{S\backslash NP/NP}$	$\lambda x.\lambda y.P(y,x)$

Open class words are assumed to belong to all families with the parts of speech suggested by the morphological analyzer, and are given every category in those families. Thus, the morphological entries in (7b) would receive all categories for all families marked with the V part of speech. The families in (8) would yield the categories in (7c) plus the corresponding transitive categories.

In a sense, this allows the hand-built lexicon alone to achieve wide coverage, but at the price of high ambiguity and over-generation. For example, the word **devoured** will receive the same categories as **walked** even though it can only appear in a transitive context. However, this lexicon was never intended to be used on its own. In §5, we show how we further restrict this lexicon using the acquired lexicon discussed in §4.

4 The Acquired Lexicon

In this section, we briefly describe how categorial lexicons can be acquired from the University of Pennsylvania Treebank (Marcus *et al.*, 1993). We then analyze a lexicon which has been extracted from sections 02-21 of the Wall Street Journal sub-corpus of the Treebank and compare it to a lexicon extracted from the Brown sub-corpus. See Hockenmaier (2000) for a more detailed description.

The Wall Street Journal sub-corpus of the UPenn Treebank contains 1 million words of parsed and tagged Wall Street Journal text collected in 1989. The Brown sub-corpus contains approximately 395,000 words of lore and fiction. Constituents are enclosed in brackets, with a label indicating the part of speech tag or syntactic category. A typical example is shown in (9).

(9)
```
(S (PP-TMP (IN In)
       (NP (DT the) (NN past) (NN decade)))
   (, ,)
   (NP-SBJ (JJ Japanese) (NNS manufacturers))
   (VP (VBD concentrated)
     (PP-CLR (IN on)
       (NP (NP (JJ domestic) (NN production))
           (PP (IN for)
               (NP (NN export))))))
   (. .) )
```

The markup distinguishes between complements and adjuncts – a feature which is crucial in our algorithm. Though the Treebank does not indicate syntactic heads, Collins (1998) gives a deterministic procedure for identifying them, and we use a slightly modified version of his heuristic. Additionally, there are different types of null elements encoding traces, multiple attachments and attachment ambiguities. The presence of these null elements allows us to infer the correct categories even in relative clauses, wh-questions and coordination constructions.

To acquire the lexicon, we have developed an algorithm which annotates the nodes in the trees with CCG categories. The resulting categories for the leaf nodes constitute the lexicon. Our algorithm is very similar in structure to Xia (1999), who extracts TAG trees from the Treebank. What we believe to be different in our approach is the extensive treatment of the various kinds of null elements. Xia (1999) uses a filter to eliminate invalid TAG trees. By definition, our algorithm always produces a set of valid lexical categories. However, our algorithm does not process any sentences which contain or are fragments, since we currently lack an adequate linguistic analysis for them.

We assume the atomic categories S, N and NP, as well as PP and ADVP, and employ features to distinguish between declarative sentences, wh-questions, yes-no questions, embedded declaratives and embedded questions. We also distinguish different kinds of verb phrases (S\NP), such as bare infinitives, to-infinitives, past participles in normal past tense, present participles, and past participles in passive VPs. This information is encoded as an atomic feature on the category. The main purpose of these features is to specify subcategorization information—for instance, the infinitival particle **to** takes a bare infinitive as its argument and yields a to-infinitive.

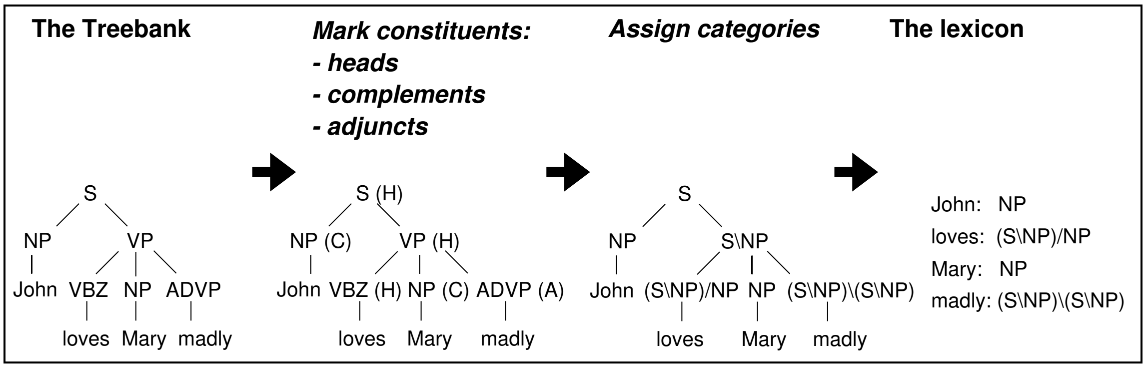


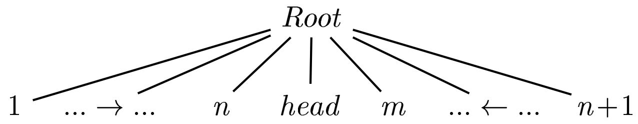
Figure 1: Acquisition of a lexicon from the Treebank

4.1 The Basic Algorithm

We will first demonstrate the basic algorithm, and then take the examples of relative clauses, right-node raising and passives to demonstrate how different kinds of null elements are treated. Figure 1 gives the outline of the algorithm. We process the trees in the corpus by first determining the constituent type of each node and then assigning categories to the nodes in the trees, depending on their constituent type and Treebank label. Once we have processed the entire corpus, we create the lexicon from the word-category pairs observed on the leaf nodes in the corpus. Additionally, lexical probabilities of the form $P(\text{category} \mid \text{word})$ or $P(\text{word} \mid \text{category})$ can be computed.

If there are no null elements, we use a simple recursive top-down procedure to annotate the nodes in a tree with the appropriate categories. We traverse each local tree in the following canonical, outside-inside order: first, we assign categories to the the children to the left of the head going from left to right, then we assign categories to the children to the right of the head, going from right to left:

(10)



This corresponds to a reverse derivation which respects the canonical order in which we traverse the tree. As we do this, we build up the category of the head child in the following manner: the innermost result category of the head child is the category of the parent. Complements to the left of the head child are added (in the order of the tree traversal) as backward arguments, complements to the right as forward arguments. Consider the example in Figure 1. Since the head child (the `VP`-node) has one complement sister with category NP to its left and its parent is S, it receives the category S\NP. The `VBZ`-node has one complement sister, NP, to its right and a parent with category S\NP. Hence it receives category (S\NP)/NP.

The category of a complement child is defined by a mapping from Treebank labels to the atomic categories. In the case of our example in Figure 1, the complements **John** and **Mary** are given the category NP.

The category of an adjunct node is a unary functor X/X or X\X, whose argument and result category are identical. The directionality of the argument is determined by the relative position of the adjunct: if it appears to the left of the head child, it is a forward looking functor, otherwise it is a backward looking functor. The category X is determined by the current head category. If we used a grammar formalism without composition, X would have to be equal to the current head category, but in the case of adjuncts of adjuncts, this would lead to a proliferation of categories. However, English CCG includes forward non-crossing composition (4a), backward non-crossing (4c) and backward crossing composition (4d), which means that we can use the following strategy: in the case of forward-looking adjuncts, X is the current head category minus all outermost forward-looking arguments. Note that we cannot strip off any backward-looking arguments, since this is equivalent to forward crossing composition, which is not allowed in English.[3] X/X can then combine with the current head category through (generalized) forward non-crossing composition. In the case of backward-looking arguments, X is the current head category minus all outermost arguments of the same directionality as the last argument— that is, if the outermost argument of the current head category is forward-looking, then we can strip off all outermost forward arguments (corresponding to generalized backward crossing composition). If the outermost argument is backward-looking, we can strip off all outermost backward arguments (generalized backward non-crossing composition). In the case of VP-adjuncts, however, we stipulate that we do not generalize beyond the S\NP level, since we want to distinguish verbal adjuncts from sentential adjuncts. For instance, consider the adjunct **madly** in Figure 1. Since its parent's category is S\NP and it is to the right of the head verb, it receives the category (S\NP)\(S\NP). Adjunct categories do not carry any morphological features.

4.2 Dealing with Null Elements

Unbounded dependencies The Treebank represents wh-questions, relative clauses, topicalization of complements, tough movement and parasitic gaps in terms of movement. The "moved" constituent is co-indexed with a trace (*T*) which is inserted at the extraction site, as in (11).

(11)
```
(NP-SBJ (NP (DT The) (NN woman))
   (SBAR (WHNP-1 (WDT who))
         (S (NP-SBJ (NNP John))
            (VP (VBZ loves)
                (NP (-NONE-  *T*-1)))
                    (ADVP (ADV madly)))))
```

CCG has a different, but similarly uniform treatment of these constructions. What in transformational terms is described as the moved constituent is analyzed in CCG as a functor over a sentence missing a complement. For instance, the relative pronoun in the following examples has the category (NP\NP)/(S/NP), while the verbs **loves** and **sent** maintain their respective canonical categories (S\NP)/NP and ((S\NP)/PP)/NP:

(12) a. the woman who John loves

b. the present which John sent to Mary

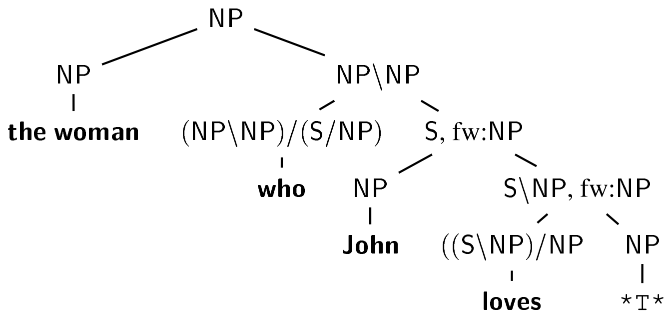
CCG allows the subject noun phrase and the incomplete verb phrase to combine via type-raising and forward composition to form a constituent with the category S/NP, which can in turn be taken as an argument of the relative pronoun. This treatment of "movement" in terms of functors over "incomplete" constituents allows CCG to keep the same category for the verb even when its arguments are extracted.

We can use the null elements in the Treebank to infer the correct categories for the verb and the moved constituent in the following way: within its local tree, the null element is treated as a normal

[3] We are following a strict interpretation of the $-convention in Steedman (2000).

constituent. This causes the category of the verb to be the same as in the non-extracted case. Then a trace (of type ‘forward’ or ‘backward’, depending on the null element’s position relative to the head child) is percolated to the top of the clause. The category of the relative pronoun is then a functor whose result is its parent category and whose argument is the category of the complement node looking for the trace in the specified direction. This results in (13).

(13)



Verb Coordination and Right Node Raising Because of CCG’s combinatory rules, no new lexical categories are required in CCG to derive the following sentences:

(14) a. Mary requests and prefers corduroy.
b. Mary prefers and I detest corduroy.

The Treebank analyzes these constructions in terms of multiple traces with the same antecedent. The constituent which is interpreted simultaneously in both conjuncts is attached at the same level as the conjuncts and is co-indexed with two *RNR*-traces in both conjuncts:

(15)
```
((S (NP-SBJ (PRP she))
    (VP (VP (VBD applied)
            (PP-CLR (IN for)
                (NP (-NONE- *RNR*-1))))
        (CC and)
        (VP (VBD won)
            (NP (-NONE- *RNR*-1)))
        (NP-1 (NN bonus) (NN pay))
        (PP-LOC (IN under)
            (NP (DT the) (NN reform) (NN law))))
    (. .)))
```

In order to obtain the correct categories for both verbs, we must take the traces, but not their antecedent into account. A postprocessing step is required, in which antecedents of *RNR*-traces are assigned the categories of the traces.

Passive VPs The subject of a passive sentence is co-indexed with a * null element which appears in the direct object position after the past participle, (16).

(16)
```
(S (NP-SBJ-1 John)
   (VP (VBD was)
       (VP (VBN hit)
           (NP (-NONE- *-1))
           (PP (IN by)
               (NP-LGS (DT a) (NN  ball))))))
```

In this case, the null element does not indicate an argument which should be reflected in the category of the participial. In the case of verbs like **pay for**, which subcategorize for a PP, the null element appears within the PP, (17).

(17)
```
(S (NP-SBJ-30 (PRP$ Its)
      (NN budget))
   (VP (VBZ is)
       (VP (VBN paid)
           (PP-CLR (IN for)
                   (NP (-NONE- *-30)))
           (PP (IN by)
                   (NP-LGS (PRP you)))))
           (. .)))
```

In this example, we have the lexical entries **paid** $\vdash$ $\mathsf{VP_{pass}/(PP/NP)}$ and **for** $\vdash$ $\mathsf{PP/NP}$. That is, the null element must be taken into account as an argument to the preposition, but it must also be percolated up to the PP level in order to assign the correct category to the participle.

4.3 Coverage of the Acquired Lexicon

We extracted a lexicon from sections 02-21 of the WSJ corpus of the Treebank. This lexicon contains 70,151 entries for 40,886 words. In total, there are 1137 category types. Figure 2 shows the frequency distribution of the category types. Out of the 1137 category types, 375 occur only once. Inspection reveals that these rare categories are mostly due to markup errors in the Treebank. On average, a word has 1.716 categories. (A lexicon acquired from the same subcorpus, but without any morphological features on verbal categories, has 499 category types. The average number of categories per word is 1.61.) Also, as can be seen from Figure 2, there are a few words which have very many categories. Most of these are closed class items, such as different inflection forms of the copula and prepositions.

Frequency distribution of category types

w/features	w/o features	Token frequency f
375	177	$f = 1$
388	203	$1 < f < 10$
236	125	$10 \leq f < 100$
84	40	$100 \leq f < 1,000$
42	27	$1,000 \leq f < 10,000$
12	15	$10,000 \leq f$

Distribution of categories for word

Words	Number of entries n
28542	$n = 1$
11865	$1 \leq n < 10$
378	$10 \leq n < 20$
50	$20 \leq n < 30$
31	$30 \leq n < 40$
20	$40 \leq n$

Figure 2: Frequency Information of Acquired Lexicon

In order to assess coverage, we extracted a new lexicon with features from section 23 alone, and compared it to the lexicon extracted from sections 02-21 (the reference lexicon), as did Xia (1999). Section 23 has 48,780 word tokens. The new lexicon has 400 category types, out of which 21 were not in the reference lexicon. Of these new category types, one occurs twice, and the rest occur only once. Again, these new types appear to be primarily caused by markup-errors in the Treebank.

For 95.16% of the word tokens in section 23, the corresponding entry existed in the reference lexicon (Xia (1999): 94.03%). 2.11% of the tokens were known words with known categories, but the exact entry was missing in the reference (Xia (1999): 3.42%). 0.045% of the tokens were known words with unseen categories (Xia (1999): 0.08%) and 2.68% of the words were unknown (Xia (1999): 2.47%– this is lower because more data is used by processing fragments). Of the unknown words, 78.01% were either N or N/N. There were no unknown words with unknown categories. If we assign N and N/N to all unknown words, we find the correct entries for 97.25% of the words. Another way of dealing with unknown words would be to back off to their POS tags, but this has not yet been implemented.

4.4 Domain-specificity of the acquired lexicon

We also extracted a lexicon from the Brown sub-corpus included in Treebank release 3. This is a corpus of general fiction and lore, and we would expect a lexicon extracted from this corpus to be substantially different from a lexicon extracted from the Wall Street Journal. The Brown corpus lexicon contains 48,310 entries for 27,163 word types (and 392,984 word tokens). On average, a word has 1.778 categories. For 88.67% of the words in the corpus, (corresponding to 43.66% of the lexical entries), the corresponding lexical entry can also be found in the lexicon extracted from sections 02-21 of the WSJ corpus. 4.83% of the tokens created new lexical entries of seen words and seen categories (accounting for 23.83% of the lexical entries). 6.26% of the tokens were pairs of previously unseen words with seen categories, and only 0.01% of the tokens were pairs of unseen words and unseen categories. Assuming that unknown words can be either N or N/N, we can find the correct lexical entry for 93.31% of the tokens in the Brown corpus in the WSJ lexicon.

In the other direction, the Brown lexicon contains the correct entries for 81.31% of the tokens in the WSJ corpus. 6.71% of the tokens in the WSJ corpus are pairs of seen words and seen categories, with the entry missing in the Brown lexicon. 0.08% of the tokens in the WSJ are pairs of seen words and unseen categories. 9.53% of the tokens which did not occur in the Brown corpus were N or N/N . Only 2.35% of the tokens were unseen words with other (previously seen) categories, and 0.003% of the tokens were unseen words with unseen categories. This means that if we assign N or N/N to unknown words, we can find the correct entry for 90.85% of the tokens in the Wall Street Journal from the Brown lexicon.

5 Supplementing the Acquired Lexicon

As discussed in §1, there have been other ambitious projects to build large CCG grammars automatically. These all share the property that their focus is on syntactic coverage with little or no attention to corresponding semantics. This is unfortunate since syntactic/semantic transparency is such an elegant aspect of the formalism. Hand-built CCG grammars can be easily designed so that every syntactic category is paired with a tailored semantic form, but, as we discussed previously, there are numerous disadvantages to building grammars by hand. The acquired lexicon helps by providing an extensive lexicon along with frequency information for word/category pairs. However, an acquired lexicon such as that presented in §4 is deficient in that it lacks semantics and morphological information.

We propose a way in which an acquired lexicon can be augmented with semantics using the lexicon discussed in §3. This information can then be used in the knowledge components of Grok to maintain discourse information.

In Grok, various modules, such as the parser, query the lexicon module for the lexical entries of a particular word. That module, in turn, queries the hand-built and acquired lexicons. These two sets of results must then be combined. The ideal case would be for the lexical entries of the hand-built lexicon to mirror those of the acquired lexicon. Then, it would be a simple matter to take the morphological features and the semantics from the hand-built lexicon to supplement the acquired one. Unfortunately, it is common for each lexicon to propose some categories that the other does not. This gives rise to three situations.

As already mentioned, the first is the simple case in which a word receives the same category from both lexicons. The categories are not exactly the same since the hand-built lexicon generally proposes significantly more morphological information. Thus, we take the result of unifying the two categories. In addition, we include the semantics from the hand-built category and the statistical information from the acquired category.

Another case is where, for a particular word, a category is specified in the hand-built lexicon but not in the acquired lexicon. It is tempting to disregard such a category as being the result of the lexicon's over-generation. However, it is possible that the category is valid and was designed to handle several different constructions simultaneously. The algorithm in §4 might not be able to acquire such a category since it only considers a single construction at a time. Fortunately, these categories almost always occur for closed class items. Thus, we employ the heuristic of taking the category if it is in a closed class family and disregarding it otherwise. Of course, no statistical information is available for these categories and further research is necessary to determine how best to provide an approximation for this. For now, we simply assign them an arbitrary, high probability.

The most difficult case is when a word has an acquired category not in the hand-built lexicon, resulting in impoverished morphology and no semantic information. One might expect that the tight connection of syntax and semantics inherent in CCG would allow one to deduce semantics directly from categories. Were it the case that each syntactic category had exactly one possible semantic category, this would indeed be trivial. But this is not the case, as shown by (18). In this simple example, **big** and **dog** are both noun modifiers. However, while **big** compositionally combines with a noun, such as in *big house*, to denote an entity which is both a house and big, *dog house* does not denote an entity which is both a house and a dog. Instead, a different semantic relation is communicated.

(18) a. **big** $\vdash \mathsf{N/N} : \lambda p.\lambda y.(p(y) \wedge \mathtt{big}(y))$

b. **dog** $\vdash \mathsf{N/N} : \lambda p.\lambda y.(p(y) \wedge \mathtt{for}(\mathtt{dog}, y))$

It seems obvious in this case that these examples can be distinguished by the fact that **dog**, in the Treebank, is annotated with the noun POS tag (`NN`) while **big** is annotated as an adjective (`JJ`). This is true, but there are many possible features that can affect these choices – syntactic arity, syntactic features, morphology, etc. Therefore, rather than attempting to hand code a mapping from syntactic categories to semantic categories, we use a machine learning approach.

We start with randomly selected entries from the acquired lexicon while maintaining the distribution of word/category pairs so that the pairs occur with the same relative frequency in our data as in the original lexicon. We exclude numbers, punctuation, and proper names as these are handled in Grok through other means (see §6). For each of these lexical entries, we produce training data consisting of the following features: the orthography of the word, its suffix, the number of syntactic arguments, various features of the acquired category, and whether the syntactic category is a verb, verb modifier, noun phrase, or noun phrase modifier. Also included is the correct general semantic form for the lexical entry. These semantic forms are logical form patterns for basic verbs, adjectives, pronouns, determiners, non-compositional noun modifiers (e.g. **dog**), prepositions, exceptive phrases (e.g. **except for**), etc. While these semantic forms contain several ambiguities akin to that in (18), it is nonetheless a rather impoverished set. Furthermore, there is no information about the semantics for categories that do not appear in the hand-built lexicon. Fortunately, it is a simple matter to add new syntactic and semantic types to the underspecified lexicon, so we believe the technique will scale.

We test both the decision tree and maximum entropy (see §6) approaches to machine learning on this task. We use Ripper (Cohen, 1995) to induce our decision tree. The training set consists of 7981 data points, and the testing set, 885 data points.

	training	testing
Baseline	86.4%	73.2%
Ripper	97.2%	97.3%
Maxent	97.8%	97.1%

Table 1: Evaluation of Inferring Semantics

As shown in Table 1, both the decision tree and the maximum entropy approaches perform in the ninety-seventh percentile at predicting the correct semantic pattern given the features described above. Since the model makes a guess in every instance, these numbers are both precision and recall. We compare these results to a baseline which simply chooses the most frequent semantic form for a word and guesses the overall most frequent semantic form if the word is unknown. As one can see, this performs significantly worse than both machine learning techniques.

6 Preprocessing

We have discussed a robust way of automatically generating a large-coverage CCG lexicon from annotated corpora. Nonetheless, free, running text, such as found in newspapers, still poses a number of problems for a CCG parser, even if it uses a lexicon which has been extracted from that very domain. Consider the following passage from the Wall Street Journal:

(19) Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29. Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.

The most obvious and universal problems are tokenization and sentence detection. Is [*Vinken,*] a single token or two? Where is the sentence boundary in *...Nov. 29. Mr. Vinken...*? Problems more specific to a system such as Grok which uses a lexicalized grammar include subsets of the unknown word problem: e.g. recognizing dates and named entities and dealing with abbreviated terms and numerical units in the text. Assigning lexical categories to such textual elements in order to process them is particularly inefficient when considering that pattern matching and statistical techniques do an excellent job of identifying them.

Our solution for these issues has been to create a pipeline of preprocessing components which use XML documents in the input/output specifications. XML (World Wide Web Consortium, 1997) provides an elegant means for structuring texts. We have built the components themselves by using maximum entropy probability models. Maximum entropy is a powerful statistical method for estimating decisions by combining diverse pieces of information. Ratnaparkhi (1998) explores the maximum entropy framework with respect to many linguistic problems and demonstrates how tools using it can achieve high accuracy and domain independence.

At present, we have built a tokenizer and sentence detector which use maximum entropy models with features based on those presented in Ratnaparkhi (1998) and Reynar and Ratnaparkhi (1997). Other tasks such as paragraph detection or dealing with figures and tables have not been implemented as yet, but the existing components have been designed to work with XML documents in a manner which permits additions such as these with minimal effort.

With tokenized and sentence detected data in hand, we are much closer to the goal of feeding the text to the CCG parser. However, there are still many things we can do to ease the parser's work and reduce the burden on the lexicon. One such task is named entity recognition. We feel that names should not be derived through standard CCG derivations for several reasons. First, the acquired lexicon is limited to those names it observes in the data. Furthermore, for a name such as **John**, the acquired lexicon produces two categories: NP for the name alone and NP/NP as in *John Smith*. This is particularly detrimental since a large factor in the efficiency of CCG parsing is dependent on lexical ambiguity and names are quite common in texts. Finally, names lack recursive structure, are not compositional, and fit a somewhat standard format, so finding a CCG derivation for them will not tell us much. We thus choose to put the burden of deriving names on another component. Then, when a sentence is given to the parser, it receives names as chunks. A side benefit of this is that we can use the results of name detection to know whether or not we can decapitalize sentence initial words so that we do not need dual lexical entries for non-names occurring at the beginning of sentences.

Our maximum entropy software, based on Ratnaparkhi (1997), allowed us to implement a simple though high performing named entity recognizer in less than a day. The results for evaluation on 20,468 training sentences and 4552 unseen testing sentences are given in Table 2. Named entity recognition is a well studied task, so it should be a simple matter to substitute a more developed approach such as Mikheev (1999), which achieves about 98.5% accuracy on unseen data and was the winning entry in MUC-7 (Mikheev *et al.*, 1998).

	training	testing
precision	96.1%	95.9%
recall	96.9%	94.8%

Table 2: Evaluation of Name Detection

This approach can be extended to other phenomena such as dates, appositives, and compound nouns. Dates can be handled similarly to names. Both nominal appositives (such as *the Dutch publishing group*) and adjectival appositives (such as *61 years old*) have a very restricted syntactic positioning and are clearly delimited by commas, so we expect another maximum entropy component would do well in detecting them. However, unlike names, appositives have a compositional meaning and can have recursive structure. Even so, at the top they are still of the category NP or NP\NP. With appositives detected, we could give the parser the appositive's lexical items with the goal NP for nominal appositives or NP\NP for adjectival appositives in order to compute its meaning before proceeding with the rest of the sentence.

Cutting up the text in the manner described above can yield significant gains for a CCG parser. The text in (19) provides an excellent example of the benefits. If we were to perform name, date, and appositive detection on that text, a cubic chart parser would visit only 25% of the cells it would using the unprocessed tokens. Furthermore, since the complexity for lexicalized grammars such as CCG is also dependent on the number of categories per word, the reduction in the number of categories for dates, numbers, and names provided by dealing with them in this way will translate into yet more efficiency gains, especially since the maximum entropy components themselves are extremely efficient. We are also interested in using finite-state transducers for other tasks like date recognition and morphological analysis.

Finally, we have implemented a part-of-speech tagger based on Ratnaparkhi (1998), and intend to extend it to "CCG"-tagging, similar to the supertagging of Srinivas (1997). The model would be trained using a CCG-annotated version of the Treebank (using the process described in §4) as data.

7 Conclusions and Future Work

The acquired lexicon specifies a set of possible categories for each word, and also includes unigram probabilities (of the form $P(\text{category} \mid \text{word})$ or $P(\text{word} \mid \text{category})$). The task of the parser is then to select the appropriate categories from this lexicon. As a baseline evaluation, we use a lexicon which includes probabilities of categories given word, and we have incorporated the following first-best search strategy into our chart parser. Each edge in the chart is given a "measure" which is the average of the probabilities of its composite categories. Edges are sorted based on the measure and processed in descending order. Termination occurs when there are no more edges or a sentence is derived for the entire string. This parser processed a list of 41 hand-created sample sentences (meant to exercise the system on various linguistic constructions) in 32.83 seconds compared to 160.73 seconds when there is no preference between edges. This is an 80% reduction in run time and it produces reasonable parses. While we are aware of the fact that speed alone is not a rigorous performance criterion, we see it as confirmation that the simple unigram probabilities discussed in §4 are already good predictors of categories during parsing. A more formal evaluation procedure would be to use this first-best parser to parse some unseen WSJ text, and compute precision and recall on the lexical categories of the highest parse against the output of our deterministic mapping from the Treebank to CCG. This would still only constitute a test of the baseline, unigram probabilities. However, one of our long-term goals is statistical parsing with CCG, and in order to do this, we need a statistical model, such as Collins (1998), which includes lexical head-dependency probabilities. We believe that our current work on extracting categorial lexicons from the Treebank is an important first step towards this goal.

We have furthermore shown how we provide additional robustness and coverage to a hand-built symbolic NLP system by using statistical information gathered from an annotated corpus. We have shown how the automatically acquired syntactic information can be combined with semantic information in the original system using machine learning techniques. We also have described how well-understood statistical preprocessing methods can be used in new ways to improve the efficiency of parsing. We feel that these ideas will provide a general framework for expanding the capabilities of symbolic grammars so that they can be used in efficient real-world applications, countering claims that formalisms such as categorial grammar are impractical and difficult to scale. On the contrary, it is precisely the properties of that very formalism, namely its high level of lexicalization and the compactness of the lexicon it allows, which make even naive approaches to parsing work reasonably well.

The methodology of reducing complexity through preprocessing steps is generally applicable to the processing of rule-based formalisms. Since Grok is a module library, its components can be glued together to make systems with different capabilities and properties. For example, the preprocessing subsystem can be used completely independently of the parsing subsystem and could be used as a front-end to another system such as XTAG. The components are also language independent; for example, a new maximum entropy model for Portuguese POS-tagging could easily be used in place of the default English model. Grok and our maximum entropy package are both distributed under the General Public License (Free Software Foundation, 1991), so anyone may freely use the techniques and tools described here, which are available for download at <`http://grok.sourceforge.net`> and <`http://maxent.sourceforge.net`>.

8 Acknowledgements

We would like to thank Mark Steedman, Chris Brew, Geert-Jan Kruijff, Ambrose Nankivell, and three anonymous reviewers for their helpful comments and suggestions.

References

Ades, A. E. and Steedman, M. J. (1982). On the order of words. *Linguistics and Philosophy*, **4**, 517–558.

Ajdukiewicz, K. (1935). Die syntaktische konnexität. In S. McCall, editor, *Polish Logic 1920-1939*. Oxford University Press, 207-231. translated from *Studia Philosophica*, 1, 1-27.

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

Cohen, W. (1995). Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth Int'l Conference*.

Collins, M. (1998). *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Curry, H. B. and Feys, R. (1958). *Combinatory Logic: Vol I*. North Holland, Amsterdam.

Doran, C. and Srinivas, B. (to appear). Developing a wide-coverage CCG system. In *CSLI Lecture Notes*. Cambridge University Press.

Free Software Foundation (1991). GNU General Public License. http://www.gnu.org/copyleft/gpl.html.

Grover, C., Carroll, J., , and Briscoe, T. (1993). The Alvey Natural Language Tools grammar. Technical report, The University of Edinburgh, Human Communication Research Centre.

Hockenmaier, J. (2000). Extraction of categorial lexica from an annotated corpus. Technical report, Division of Informatics, University of Edinburgh.

Joshi, A. (1988). Tree Adjoining Grammars. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Parsing*, pages 206–250. Cambridge University Press, Cambridge.

Joshi, A., K.Vijay-Shanker, and Weir, D. (1991). The convergence of mildly context-sensitive formalisms. In P. Sells, S. Shieber, and T. Wasow, editors, *Processing of Linguistic Structure*, pages 31–81. MIT Press, Cambridge MA.

Kaplan, R. and Bresnan, J. (1982). Lexical-Functional Grammar: A formal system for grammatical representation. In *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, MA.

Kuhn, J., Eckle-Kohler, J., and Rohrer, C. (1998). Lexicon acquisition with and for symbolic NLP-systems – a bootstrapping approach. In *Proceedings of the First International Conference on Language Resources and Evaluation (LREC98)*, Granada, Spain.

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

Mikheev, A. (1999). A knowledge-free method for capitalized word disambiguation. In *37th Annual Meeting of the Association for Compuational Linguistics*, pages 159–166.

Mikheev, A., Grover, C., and Moens, M. (1998). Description of the LTG system used for MUC-7. In *Proceedings of 7th Message Understanding Conference (MUC-7)*.

Pollard, C. and Sag, I. (1994). *Head Driven Phrase Structure Grammar*. CSLI/Chicago University Press, Chicago.

Ratnaparkhi, A. (1997). A simple introduction to maximum entropy models for natural language processing. Technical Report IRCS-97-08, University of Pennsylvania, Institute for Research in Cognitive Science.

Ratnaparkhi, A. (1998). *Maximum Entropy Models for Natural Language Ambiguity Resolution*. Ph.D. thesis, University of Pennsylvania.

Reynar, J. and Ratnaparkhi, A. (1997). A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the International Conference on Spoken Language Processing*, pages 16–19, Washington D.C.

Srinivas, B. (1997). *Complexity of Lexical Descriptions and Its Relevance to Partial Parsing*. Ph.D. thesis, University of Pennsylvania. IRCS Report 97-10.

Steedman, M. (1985). Dependency and coordination in the grammar of dutch and english. *Language*, **61**, 523–568.

Steedman, M. (1987). Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, **5**, 403–439.

Steedman, M. (1996). *Surface Structure and Interpretation*. MIT Press, Cambridge Mass. Linguistic Inquiry Monograph, 30.

Steedman, M. (2000). *The Syntactic Process*. The MIT Press, Cambridge Mass.

van Genabith, J., Way, A., and Sadler, L. (1999). Semi-automatic generation of F-structures from Treebanks. In M. Butt and T. H. King, editors, *Proceedings of the LFG99 Conference*. CLSI Publication.

Vijay-Shanker, K. and Weir, D. (1994). The equivalence of four extensions of context-free grammar. *Mathematical Systems Theory*, **27**, 511–546.

Villavicencio, A. (1997). *Building a wide-coverage Combinatory Categorial Grammar*. Master's thesis, Cambridge.

World Wide Web Consortium (1997). Extensible markup language (xml). Web Page. http://www.w3.org/XML/.

Xia, F. (1999). Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium(NLPRS-99)*.

XTAG-group (1999). A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS-98-18, University of Pennsylvania.