Fully Lexicalising CCGbank with Hat Categories

Matthew Honnibal and **James R. Curran**
School of Information Technologies
University of Sydney
NSW 2006, Australia
{mhonn,james}@it.usyd.edu.au

Abstract

We introduce an extension to CCG that allows form and function to be represented simultaneously, reducing the proliferation of modifier categories seen in standard CCG analyses.

We can then remove the non-combinatory rules CCGbank uses to address this problem, producing a grammar that is fully lexicalised and far less ambiguous.

There are intrinsic benefits to full lexicalisation, such as semantic transparency and simpler domain adaptation. The clearest advantage is a 52-88% improvement in parse speeds, which comes with only a small reduction in accuracy.

1 Introduction

Deep grammars return parses that map transparently to semantic analyses, allowing information extraction systems to deal directly with content representations. Usually, this mapping is lexically specified, by linking lexical entries to semantic analyses. This property, lexicalisation, is central to some of the linguistic theories behind deep grammars, particularly Combinatory Categorial Grammar (Steedman, 2000) and Lexicalised Tree Adjoining Grammar (Joshi, 1999).

Lexicalisation can also help deep grammars achieve satisfactory parse times. Lexicalised grammars use few rules, which simply manipulate the lexical categories. The categories can be quickly assigned in a supertagging pre-process, dramatically reducing the search space the parser must explore (Bangalore and Joshi, 1999).

Combinatory Categorial Grammar (CCG) is well suited to this strategy, and Clark and Curran (2007) have highlighted the division of labour between the parser and the supertagger as one of the critical aspects of their approach to statistical CCG parsing. In their system, the division is managed with parameters that control how many categories the parser's chart is seeded with. But even if the parser is only given one category per word, it still has a lot of freedom — because the grammar it uses is not fully lexicalised.

In a fully lexicalised CCG grammar, modifier categories refer to the category of their head. This category does not necessarily represent the head's constituent type. For instance, the category of an adverb like *still* depends on whether it is modifying a predicate verb (1), or a clausal adjunct (2):

(1)	*The lion*	*was*	*lying*	*still*
	NP	VP/VP	*VP*	$VP\backslash VP$
(2)	*The lion*	*waited,*	*lying*	*still*
	NP	*VP*	$VP\backslash VP$	$(VP\backslash VP)\backslash(VP\backslash VP)$

Analyses like these are problematic because the training data is unlikely to include examples of each word in every syntactic environment that requires a new category. Hockenmaier and Steedman's (2007) solution was to add category specific phrase-structure rules to the grammar, which disrupts the linguistic principles of the formalism, and introduces over-generation and ambiguity as shown in Figure 1.

This paper proposes a new way to balance lexical and grammatical ambiguity in CCG. We introduce an extension to the formalism that allows type-changing rules to be lexically specified. The extension adds a new field to the category objects, and one additional rule to utilise it. This allows the formalism to express type-changing operations in a theoretically desirable way.

Lexically specifying the type-changing rules reduces the ambiguity in the grammar substantially, which leads to substantial improvements in parsing efficiency. After modifying the C&C parser and CCGbank, the parser runs almost twice as quickly, with only a 0.5% reduction in accuracy.

Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, pages 1212–1221,
Singapore, 6-7 August 2009.

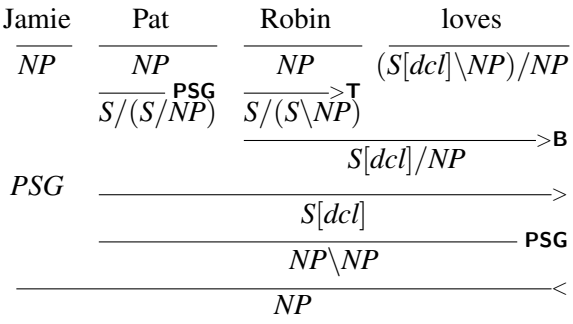


Figure 1: Over-generation by CCGbank rules.

2 Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG) (Steedman, 2000) is a lexicalised grammar formalism based on categorial grammar (Bar-Hillel, 1953). CCG can be distinguished from other CG extensions, such as categorial type-logic (Moortgat, 1997) by its attention to linguistic minimalism. One aim of the theory is to explain universal constraints on natural language syntax, so the generative power of the formalism is intended to closely match what natural language seems to require.

Steedman and Baldridge (2007) argue that the requirements can be fulfilled almost entirely by two basic rule types: application and composition. Direction specific instances of these types yields a grammar that consists of just six rules.

Initially, it seemed that some of the rules had to be restricted to certain contexts, particularly in languages that did not allow scrambling. Baldridge and Kruijff (2003) have since shown that rules could be restricted lexically, using a hierarchy of slash subtypes. This relieved the need for any language specific meta-rules, allowing CCG to offer a completely universal grammar, and therefore a theory of the innate human language faculty.

With a universal grammar, language specific variation is confined to the lexicon. A CCG lexical category is either an atomic type, like N, or a function that specifies an argument in a particular direction, and a result, like $S\backslash NP$ (where S is the result, NP the argument, and $\backslash$ indicates the argument must be found to the left).

Hockenmaier and Steedman (2007) showed that a CCG corpus could be created by adapting the Penn Treebank (Marcus et al., 1993). CCGbank has since been used to train fast and accurate CCG parsers (Clark and Curran, 2007).

3 The Need for Type-changing in CCG

We argue that there is a clear need for some sort of type-changing mechanism in CCG. The practical need for this has been known since at least Hockenmaier (2003), who introduced a type-changing mechanism into CCGbank in order to control the problem referred to as *modifier category proliferation*. We briefly describe the problem, and then the prominent solutions that have been proposed.

Unlike formalisms like LTAG and HPSG, CCG does not use different grammatical rules for arguments and adjuncts. Instead, modifier categories take the form $X_1|X_1$, where X is the category of the constituent being modified, and the subscript indicates that the result should inherit from the argument via unification. The modifier can then use the application rule to attach to its head, and return the head unchanged:

(3) *unusually* $(S[adj]\backslash NP)/(S[adj]\backslash NP)$ *resilient* $S[adj]\backslash NP$

unusually here modifies the predicative adjective *resilient*, attaching it as an argument using forward application. This prevents *resilient* from having to subcategorise for adjuncts, since they are optional. The problem is that *unusually* must subcategorise for the function of its head. If *resilient* changes function and becomes a noun modifier, its modifiers must change category too:

(4) *an* NP/N *unusually* $(N/N)/(N/N)$ *resilient* N/N *strain* N

There is often a way to analyse around the need for type-changing operations in CCG. However, these solutions tend to cause new difficulties, and the resulting category ambiguity is quite problematic (Hockenmaier and Steedman, 2002). The fact is that form-to-function coercions are quite common in English, so the grammar needs a way to have a constituent be modified according to its form, before undergoing a type-change to its function category.

One way to describe the problem is to say that CCG categories have an over-extended *domain of locality* (Joshi, 1999), the part of the derivation that it describes. A category should specify all and only the dependencies it governs, but CCG modifier categories are often forced to specify their heads' dependencies as well. These undesirable notational dependencies can also prevent modifier categories from factoring recursion away from their domain of locality.

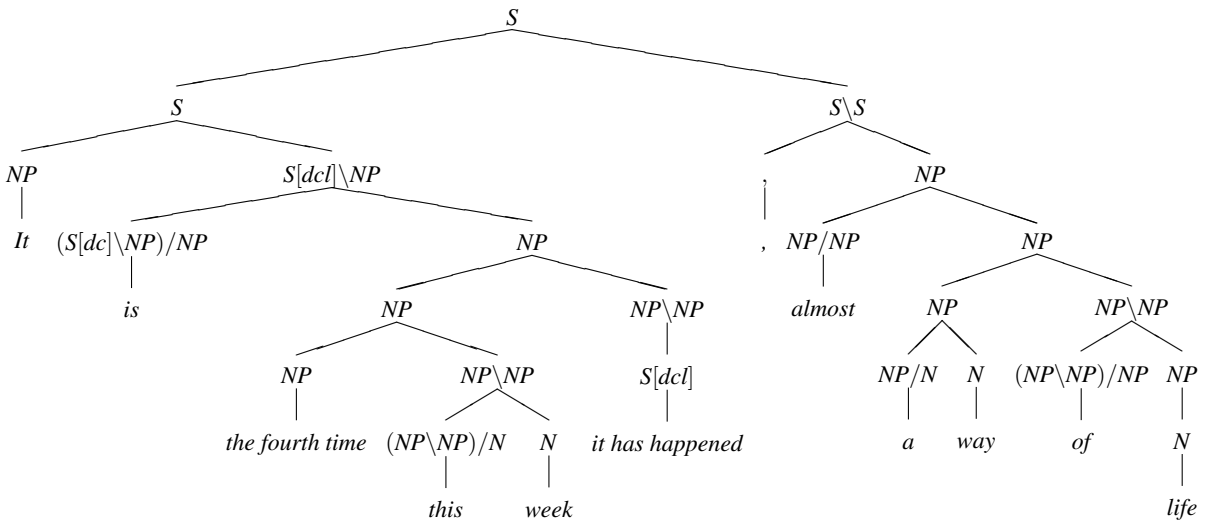


Figure 2: CCGbank derivation showing PSG rules.

4 Problems with Existing Proposals

This section completes the motivation of the paper by arguing that the existing proposals for type-changing are linguistically unsatisfactory, practically difficult, or a combination of the two.

4.1 Problems with PSG Rules

Hockenmaier and Steedman (2002) includes a brief discussion of the modifier category proliferation problem, and introduces unary phrase-structure rules to address the situation. Figure 2 shows two such rules. The $<S[dcl] \rightarrow NP\backslash NP>$[1] rule allows the reduced relative clause, *it has happened*, to be analysed as a modifier without affecting the category any modifiers that might attach to it. The other PSG type-changing rule in the derivation, $<, NP \rightarrow S\backslash S>$ enables the extraposition, using the punctuation to make the rule more precise.

One alternative to type-changing rules here would be to have *time* subcategorise for the clause, with a category like $NP/S[dcl]$. This would capture the constraint that only a restricted subset of nouns can be extracted as adjuncts in this way. The problem is that the extra argument would interfere with the attachment of adjuncts like *this week* to the *NP*, because the $NP\backslash NP$ category cannot be allowed to participate in backwards cross-composition rules (Baldridge and Kruijff, 2003).

There are 204 type-changing PSG rules in the training partition of CCGbank. 53 of the frequent rules transform produce modifier categories, 48 of them transforming verbal categories. The PSG rules also handle a variety of other constructions, such as form/function discrepancies like gerund nominals. By far the most frequent rule, with 115,333 occurrences, is $<N \rightarrow NP>$, which transforms bare nominals into noun phrases.

Steedman and Baldridge (2007) describes the CCG grammar as consisting of just 6 language universal combinatory rules, plus two lexical operations (type raising). Not only do the 204 category specific type-changing rules in CCGbank make the grammar ambiguous, they also run contrary to the design principles of the formalism.

CCG is a linguistically motivated formalism, which means it is not only interested in providing a convenient, computable notation for grammar development. In addition, it constitutes a hypothesis about the nature of the human language faculty. Like other lexicalised formalisms, part of the theory is that it is the grammar that is innate, and the lexicon is acquired.

If the grammar is innate, it must be language universal, confining all language specific variation to the lexicon. Baldridge and Kruijff (2003) described how the remaining language specific grammatical constraints described by Steedman (2000) could be controlled in the lexicon, using multi-modal slashes that have since become integrated into the main body of the theory (Steedman and Baldridge, 2007).

In addition to being linguistically undesirable, the PSG rules in CCGbank produce practical difficulties. Every additional rule increases ambiguity, motivating the C&C system to choose to implement only the most frequent. This decreases

[1] Phrase-structure rules are presented in bottom-up notation.

the parser's coverage, and introduces another dimension of domain sensitivity. For instance, the type-changing rule that allows gerund nominals, $<S[ng]\backslash NP \rightarrow NP>$, occurs roughly 300 times in the training data. The parser does not implement this rule, so if it is ported to a new domain, where the construction is frequent, the rule will have to be added. Presumably, the parser would also benefit from the removal of rules which are infrequent in some new, target domain.

The restricted set of PSG rules the parser does implement results in considerable added ambiguity to the grammar. Figure 1 shows how the rules interact to produce over-generation.

The PSG rules are also a barrier to the semantic transparency of the theory, one of its most attractive properties for natural language engineering. CCG derivations are isomorphic to semantic analyses, because the derivation instantiates dependencies between CCG categories that can be paired with semantic categories. This isomorphism is disrupted by the addition of PSG rules, since the grammar is no longer lexicalised. Often, the rules can be semantically annotated, restoring the isomorphism; but sometimes, this cannot be done.

For instance, the extraposition rule in Figure 2 transforms the *NP* category into $S\backslash S$. There is no syntactic argument on the *NP* category to map the dependency to, so the dependency cannot be created (and is in fact missing from CCGbank).

4.2 Lexical Rules and Zero Morphemes

The CCGbank PSG extension is closely related to the zero morpheme categories proposed by Aone and Wittenburg (1990), which they suggest be compiled into unary type-changing rules for processing. At first glance, it seems that conceptualising the rules as zero morphemes offers a way to locate them in the lexicon, avoiding the linguistic difficulties of having a language-specific grammar. However, CCG aims to provide a transparent interface between the surface form and the semantic analysis, so epsilon categories, traces, movement rules and other unrealised structures are explicitly banned (Steedman, 2000).

From a processing standpoint, if zero morpheme categories are not compiled into phrase-structure rules, then they will complicate the category assignment phase considerably, since we can no longer assume that exactly one category will be assigned per word. We are not aware of any proposal for how this difficulty might be overcome.

Carpenter (1992) provides a different suggestion for how sparse modifier categories can be accommodated. His solution is to use meta-rules that systematically expand the lexicon, much like the lexical rules used in HPSG (Flickinger, 1987), which exploit structural regularities to ensure that the lexicon is more complete.

The problem with this is that it does not actually make the category set less sparse, so the supertagger's task is just as difficult. The only advantage is that its dictionary will be more complete. This is important, but does not solve the underlying inefficiency in the grammar: CCG categories have an over-extended domain of locality, because they cannot represent form and function simultaneously. This is why some type-changing mechanism is required.

5 Lexically Specified Type-Changing

This section describes our mechanism for lexically specifying the PSG rules in CCGbank. Figure 3 shows an example of a reduced relative clause analysed using our extension, *hat categories*.

CCGbank deploys a solution that achieves *form transparency* at the expense of *type transparency*, by allowing type-changing rules that are not lexically specified. One way to recover the lost type transparency would be to demand that lexical categories specify what type changing rule (if any) the category can eventually undergo. For instance, imagine we have two type-changing rules we wish to include in our grammar:

a) $S[ng]\backslash NP \Rightarrow NP\backslash NP$

b) $S[ng]\backslash NP \Rightarrow VP\backslash VP$[2]

With these two rules, there will be three ways the $S[ng]\backslash NP$ category might behave in a derivation. What we need are two extra categories to control this:

1. $S[ng]\backslash NP$ only allows combinatory rules.
2. $(S[ng]\backslash NP)^a$ allows rule a, but not rule b.
3. $(S[ng]\backslash NP)^b$ allows rule b, but not rule a.

Instead of encoding a reference to a rule, we encode the production rule itself in the category.

[2] $S\backslash NP$ is occasionally abbreviated as *VP*.

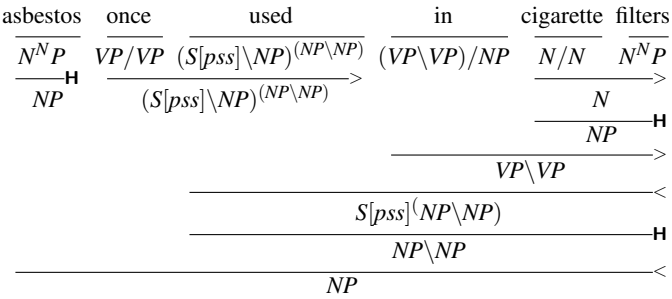


Figure 3: Analysis of a reduced relative clause with lexicalised type-changing.

$$\begin{bmatrix} \text{CAT} & (S[ng]\backslash NP)^{NP\backslash NP}/NP \\ \text{RES} & \begin{bmatrix} \text{CAT} & (S[ng]\backslash NP)^{NP\backslash NP} \\ \text{RES} & \begin{bmatrix} \text{CAT} & S \\ \text{FEAT} & ng \end{bmatrix} \\ \text{ARG} & NP_1 \\ \text{DIR} & \backslash \\ \text{HAT} & NP\backslash NP_1 \end{bmatrix} \\ \text{ARG} & NP \\ \text{DIR} & / \\ \text{HAT} & [\text{-}] \end{bmatrix}$$

Figure 4: AVM of a hat category.

This allows us to remove the rule from the grammar. Since the bottom of the production will always be the category itself, we can just specify how the category can be rewritten:

1. $S[ng]\backslash NP$ can be combined, but not rewritten.
2. $(S[ng]\backslash NP)^{NP\backslash NP}$ can be rewritten as $NP\backslash NP$.
3. $(S[ng]\backslash NP)^{VP\backslash VP}$ can be rewritten as $VP\backslash VP$.

We refer to the superscript category as a *hat* category, as a reference to the notation, but also to denote the fact that it allows the category to perform a different function, or put a different 'hat' on. Categories that have a hat specified are referred to as *hatted* categories.

5.1 Changes to Category Objects

Figure 4 shows an AVM representation of the $(S[ng]\backslash NP)^{NP\backslash NP}/NP$ category. A field, labelled *hat*, has been added to store the destination category of the result, $NP\backslash NP$. The NP argument in the hat category is co-indexed with the NP argument in the hatted category. The NP argument is also co indexed with the result of the destination category, reflecting the fact that the $NP\backslash NP$ category is a modifier, whose head will be the head of its argument.

Hat categories are handled the same as any other field during unification. If the two hat fields cannot be unified, unification fails; and if one hat field has an empty value, it inherits the value of the hat field of the other category when unification succeeds. CCG already requires a unification process for agreement features (Hockenmaier and Steedman, 2007); the hat categories we have introduced behave identically.

As Figure 4 shows, hat categories can be added to inner results, allowing arguments to be applied before the type-changing rule. We add a restriction that prevents categories with an outermost hat field from applying arguments — essentially equivalent to stipulating that the slash in a category like $S[ng]\backslash NP$ must have a null mode.

We also stipulate that only adjuncts may apply hatted arguments, which can also be lexically represented by assuming that all non-adjunct categories have a null value in their hat field, causing unification with a hatted category to fail.

Together, these restrictions ensure that the unary rule is used. The hatted category cannot function as a non-hatted category, because it cannot use its own arguments, and cannot be used as an argument of another category. This prevents hat categories from forming categories that are functionally *disjunctive*: the notation cannot be used to simulate something like an optional argument.

5.2 The Type-Change Rule

To replace the 204 PSG rules in CCGbank, we only need to introduce one extra schematic rule into the grammar:

$$X^Y \Rightarrow Y \tag{5}$$

This rule simply unpacks the category, performing the lexically specified type-change.

5.3 Generative Power

Because hat fields are only transmitted when categories are successfully unified, there is no way to produce a novel $X \Rightarrow Y$ unary production during a derivation. This means that any derivation that can be produced using the schematic type-change rule we have added to the grammar can be produced by adding a set of unary phrase-structure rules instead — ensuring that we do not require any extra generative power than is required to parse CCGbank.

The hat categories do increase the strong generative power of a CCG grammar that does not include the CCGbank type-changing rules. We suggest that this is desirable, in line with Joshi's (1999) argument that formalisms should be designed to have the maximum expressivity while maintaining the minimum weak generative power necessary to produce the constructions that have been observed in natural language.

6 Lexicalising Type-raising

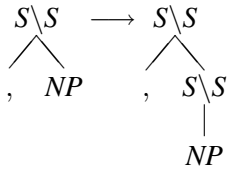
So far, we have focused on replacing the phrase-structure rules added to CCGbank, which are not part of the CCG linguistic theory. However, the theory does include some type-changing rules, referred to as *type-raising*. Forward and backward type-raising are used to transform a category X into the logically equivalent categories $T/(T\backslash X)$ and $T\backslash(T/X)$ respectively.

Type-raising is generally described as a lexical operation, rather than a grammatical rule, because only certain language specific combinations of T and X produce valid type-raise categories. However, no specific mechanism for controlling type-raising has been proposed.

Hat categories are an obvious candidate for this, so we perform an additional set of experiments which lexicalise the type-raising rules in CCGbank, in addition to the PSG rules.

7 Adapting CCGbank

This section describes how we converted CCGbank's PSG rules into analyses that used hat categories. Most of the PSG rules are unary, which meant that our changes were limited to adding hat categories to the child of the unary production and its subtree. The binary PSG rules that we converted effectively just used punctuation as a cue for a unary type-change, as seen in the extraposition rule in Figure 2. These were handled by adding an extra node for the punctuation application, leaving a unary production:



(6)

An alternative analysis would be to assign the punctuation mark a category to perform the type-change — in this case, $(S\backslash S)/NP$. However, this analysis will be unreliable for many genres, where punctuation is used inconsistently, so we preferred that hat category analysis, which we found produced slightly better results.

We used the same method to convert cases where CCGbank used conjunctions to cue a type-change, where the Penn Treebank conversion process produced a derivation where two sides of a coordination had different categories. There were 90 such conjunction coercion rules, which we have not counted amongst the 204 PSG rules, since they are ultimately caused by conversion noise.

The main complication when adapting CCGbank was the fact that CCG node labels are interdependent through a derivation. If one node label is changed, its immediate children have to change node label too, and the changes must be propagated further from there.

Since the dependency between the parent and its two children is different for each combinator, our node change rules determine the rule used for the original production, and then invoke the appropriate replacement rule. In general, the rules find the result (A_r) and argument (A_a) of the original parent A and replace them with the appropriate part of the new parent B. If one of the children is an adjunct category, a different rule is used. The node change rules for forward combinators are:

App	A/Y	Y	$\Rightarrow$	B/Y	Y
Comp	A_r/Y	Y/A_a	$\Rightarrow$	B_r/Y	Y/B_a
Adj. app	A/A	A	$\Rightarrow$	B/B	B
Adj. comp	A_r/A_r	A_r/A_a	$\Rightarrow$	B_r/B_r	B_r/B_a

The translation rules for backward and crossed combinators are directly analogous, with the slashes permuted appropriately.

8 Adapting the CCG Parser

We took the standard 1.02 release of the C&C parser Clark and Curran (2007) and implemented the changes required for lexically specified type-changing.

	Section 00							Section 23						
	LP	LR	LF	LF_{auto}	sent	cat	cov	LP	LR	LF	LF_{auto}	sent	cat	cov
CCGbank derivs	87.18	86.31	86.74	84.78	35.15	94.04	99.06	87.76	86.99	87.38	84.84	37.03	94.26	99.63
Hat derivs	86.64	86.91	86.77	84.44	35.03	93.27	99.53	86.94	87.26	87.10	84.76	36.62	93.35	99.71
Hat+TR derivs	86.58	86.87	86.73	84.16	34.47	93.10	99.63	86.83	87.16	87.00	84.67	36.73	93.17	99.75
CCGbank hybrid	88.07	86.49	87.27	85.30	35.94	94.16	99.06	88.36	87.02	87.68	85.27	36.74	94.33	99.63
Hat hybrid	87.30	86.94	87.12	84.85	35.40	93.31	99.53	87.26	87.03	87.15	84.79	36.25	93.24	99.71
Hat+TR hybrid	85.79	85.30	85.55	83.13	31.90	92.48	99.63	85.93	85.65	85.79	83.39	32.03	92.46	99.75

Table 1: Labelled Precision, Recall and F-measure, coverage results on Section 00 and Section 23.

The most significant change was building hat passing and unification into the existing unification engine. For many parsers, this would have been straightforward since they already support unification with complex feature structures. However, one of the advantages of CCGbank is that the unification required is quite simple, which is one of the reasons why the C&C parser is very fast. We would estimate that adding hat passing doubled the complexity of the unification engine.

The second step was to add support for hat passing to all of the existing combinators, because they do not use the unification engine to construct the result category. Since each of the combinators is hard-coded for speed, this was time-consuming and error prone. However, we created a detailed set of regression tests for the new versions which greatly reduced our development time.

Finally, we needed to turn off the existing unary rules in the parser, and add the simple additional type-change rule.

9 Setting Dictionary Thresholds

The main parameterisation we performed on the development section was to tune the K parameter of the parser, which controls the frequency at which a word's *tag dictionary* is used during supertagging. For words more frequent than K, the supertagger is restricted to choosing between categories that have been assigned to the word in the training data. Otherwise, the POS dictionary is used instead. The K parameter has multiple values, because the supertagger and parser are integrated such that the supertagger initially supplies only a narrow beam of categories to the parser, which is widened if parsing fails.

Since we have made the category set larger, the default values of $K = 20, 20, 20, 20, 150$ produces poor performance, up to 1.5% lower than the figures we report in Table 1. We set the K parameter to $50, 300, 80, 80, 3000$. We investigated the effect of this setting on the original model, and found that it had little effect, so we continued using the default values for the original model.

We also experimented with altering the β values for the hat parser, which did not improve the performance of the state-of-the-art parsing models.

	Section 00		Section 23	
Training	Gold	Auto	Gold	Auto
CCGbank derivs	399	413	639	544
Hat derivs	552	566	1070	827
Hat+TR derivs	718	677	1072	906
CCGbank hybrid	369	379	564	480
Hat hybrid	505	513	921	678
Hat+TR hybrid	645	601	913	785

Table 2: Parse speeds in words per second.

	Original		Hat	
	Types	Frequency	Types	Frequency
Binary CCG	2,714	1,097,809	3,840	1,097,358
Type-raise	52	3,998	52	3,996
Unhat	0	0	241	161,069
Binary PSG	215	1,615	74	172
Unary PSG	157	159,663	0	0

Table 3: Production types and frequencies.

10 Parsing Results

The left side of Table 1 shows our performance on the development data, Section 00. All of the dependency results we report refer to the original dependencies distributed with CCGbank. To ensure our results were comparable, we produced a mapping table of dependency labels from sections 02-21, used for parser training. The table maps the dependency labels in our corpus to the most frequent label assigned to matching dependencies in CCGbank. The correct label is assigned 99.94% of the time. The hat categories move to the the lexicon information that used to be represented in the grammar, resulting in a larger, more informative

category set, making the category accuracies (the *cat* column in the table) not comparable.

We experimented with two of the parsing models described by Clark and Curran (2007). The *derivs* model uses features calculated over the derivations, while the *hybrid* model uses features calculated on the dependency structures. However, unlike the *deps* model Clark and Curran (2007) describe, the *hybrid* model uses two sets of derivation-based constraints. One set are the normal form constraints, as described by Eisner (1996). It also uses constraints that prevent the parser from using productions that were not seen in the training data. The hybrid model is slightly more accurate, but also slightly slower, because the dependency-based decoding is less efficient.

All of the systems were within 0.5% in accuracy on the development set, with one exception. The HAT+TR version performed very poorly with the hybrid model, while its performance with the derivs model was comparable to the other systems. The same drop in performance occurs on the evaluation set. We do not currently have a convincing explanation for this, but we presume it is the result of some unforeseen interaction between the removal of the type-raising rules from the grammar and the dependency-based features.

The accuracy results on the test data, Section 23, saw similar trends, except that the gap between the hat systems and the original CCGbank increased slightly. The CCGbank hybrid model was only 0.1% more accurate than the HAT hybrid model on Section 00, but is 0.5% more accurate on Section 23.

Table 2 compares the parse speeds for the lexicalised hat corpora against a parser trained on the original CCGbank, using the two models. Exactly the same settings were used to obtain parse times as were used in the accuracy experiments. The experiments were all performed on a single core 2.6GHz Pentium 4 Xeon. Speeds are reported as words parsed per second.

On both Section 00 and Section 23, with both the derivs and hybrid models, the HAT system was substantially faster than the original parser. The HAT+TR system was faster than the HAT system using automatic POS tags, and slightly faster on Section 00.

The hat categories allow quite favourable tradeoffs between speed and accuracy to be made. The original models allow us to parse with automatic POS tags at 480 words per second with 85.27% accuracy with the hybrid model, or at 544 words per second with 84.86% accuracy using the derivs model. Using the HAT derivs model, we could instead parse at 827 words per second with 84.76% accuracy, or at 906 words per second and 84.67% accuracy using the HAT+TR system.

In summary, the HAT and CCGbank derivs models are equivalent in accuracy, but the HAT version is 52% faster. The CCGbank hybrid model remains the most accurate, but there will also be many tasks where the 88% improvement in speed will make it worth using the HAT+TR derivs parser instead of the CCGbank hybrid model, at a cost of 0.6% accuracy.

11 Corpus Statistics

Table 3 shows the number of types and the number of occurrences of CCG combinatory rules and PSG rules occurred in CCGbank and the hat corpus.

The hat corpus removes almost all unlicensed productions, leaving only a long tail of rare productions that are the result of noisy derivations. These productions are generally symptomatic of problematic analyses, and are difficult to address automatically because they do not conform to any consistent pattern. We have omitted the hat+TR corpus in these figures, because it only differs from the the hat corpus with respect to type-raising productions.

Lexicalising the corpus increases the number of categories required substantially. There are 407 categories that occur 10 or more times in the training section of CCGbank. The equivalent figure for the HAT corpus is 507, and for the HAT+TR corpus it is 540.

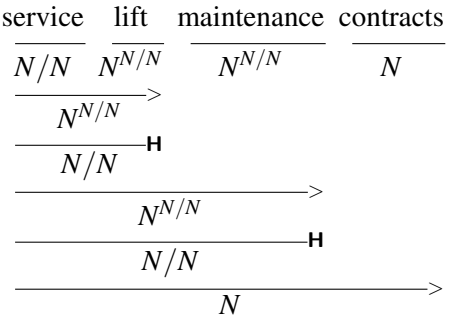
12 Cleaner Analyses with Hat Categories

The lexicalised type-changing scheme we have proposed offers many opportunities for favourable analyses, because it allows form and function to be represented simultaneously. However, we have limited our changes to replacing the existing CCGbank non-combinatory rules. This allows us to compare the two strategies for controlling modifier category proliferation more closely, but still offers some improved analyses.

The most frequent unary production in CCGbank, the $N \Rightarrow NP$ rule, ensures that nominals can always take the N category, so adjectives seldom need to be assigned NP/NP. Because ad-

jectives and nouns are open class, and bare noun phrases are fairly common, this reduction in category sparseness is quite important.

Lexicalising the type changing rule forces the head noun to acquire a different category, but does ensure that its modifiers can attach at the *N* level — which is also more linguistically desirable:



This analysis also prevents the extreme category proliferation problem caused by left-branching noun phrases:

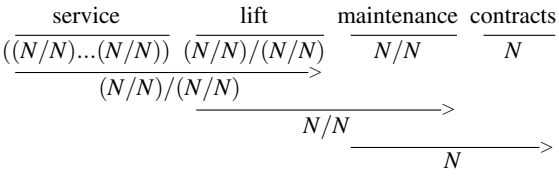


Figure 3 shows a more typical example of an improved analysis. The non-finite clause is functioning as an adnominal, but its modifier is able to select its canonical category.

One of the advantages of the CCGbank phrase-structure rules is that they allow the corpus to include derivations for which no valid CCG parse can be formed. The C&C parser has difficulty taking advantage of these extra sentences, however, because only so many of the arbitrary binary PSG rules can be added to the grammar without making it too ambiguous. Once these rules are lexicalised, the categories that produce them can be added to the lexicon as unexceptional, albeit rare, cases.

13 Conclusion

Lexicalised grammars represent most of the information in a derivation with a sequence of lexical categories. Traditional CCG analyses require redundancy between categories whenever there is nested modification, which suggests that such analyses will encounter sparse data problems.

While the addition of phrase-structure rules prevents this proliferation of modifier categories, it does so at a high price. The bulk of the type-changing rules in CCGbank are not implemented in the C&C parser, because to do so would increase the ambiguity in the grammar enormously.

CCG parsers must carefully manage ambiguity, because there are many ways to bracket the same CCG derivation. Even with a restricted set of PSG rules, the C&C parser experiences very large chart sizes. In addition to making the grammar more ambiguous, the PSG rules make it less theoretically sound, and more difficult to produce semantic analyses from the parser's output.

We have show how CCG analyses can be fully lexicalised in a way that closely mirrors the introduction of phrase-structure rules. The result is a corpus that produces faster, accurate parsers, is well suited for domain adaptation, and allows for more transparent semantic analysis. We can also use the same mechanism to lexically specify type-raising, the first concrete proposal to handle type-raising as a lexical transformation we are aware of.

From an immediate, empirical perspective, we have substantially improved the parsing speed of what is already the fastest deep parser available. Improvements in parsing efficiency are important in making parsing a practical technology, since the volume of text we have available for processing is growing even faster than the processing resources we have available.

Acknowledgements

We would like to thank Stephen Clark and the anonymous reviewers for EMNLP and the Grammar Engineering Across Frameworks workshop for their valuable feedback. This work was supported by the Australian Research Council under Discovery Project DP0665973.

References

Chinatsu Aone and Kent Wittenburg. 1990. Zero morphemes in unification-based combinatory categorial grammar. In *ACL*, pages 188–193.

Jason Baldridge and Geert-Jan Kruijff. 2003. Multi-Modal Combinatory Categorial Grammar. In *Proceedings of the European Association of Computational Linguistics (EACL)*.

Srinivas Bangalore and Aravind 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*, 29:47–58.

Bob Carpenter. 1992. *Categorial grammars, lexical rules, and the English predicative*, chapter 3. Oxford University Press.

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 *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL-96)*, pages 79–86. Santa Cruz, CA, USA.

Dan Flickinger. 1987. *Lexical Rules in the Hierarchical Lexicon*. Ph.D. thesis, Stanford University, Stanford, CA.

Julia Hockenmaier. 2003. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh.

Julia Hockenmaier and Mark Steedman. 2002. Acquiring compact lexicalized grammars from a cleaner treebank. In *Third LREC*, pages 1974–1981.

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.

Aravind K. Joshi. 1999. Explorations of a domain of locality: Lexicalized tree-adjoining grammar. In *CLIN*.

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

Michael Moortgat. 1997. Categorial type logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, chapter 2, pages 93–177. Elsevier, Amsterdam and MIT Press, Cambridge MA.

Mark Steedman. 2000. *The Syntactic Process*. The MIT Press, Cambridge, MA.

Mark Steedman and Jason Baldridge. 2007. Combinatory categorial grammar. In Robert Borsley and Kersti Borjars, editors, *Non-Transformational Syntax*. Blackwells.