Robust Conversion of CCG Derivations to Phrase Structure Trees

Jonathan K. Kummerfeld[†] **Dan Klein**[†] **James R. Curran**[‡]
[†]Computer Science Division
University of California, Berkeley
Berkeley, CA 94720, USA
{jkk,klein}@cs.berkeley.edu

[‡]ə-lab, School of IT
University of Sydney
Sydney, NSW 2006, Australia
james@it.usyd.edu.au

Abstract

We propose an improved, bottom-up method for converting CCG derivations into PTB-style phrase structure trees. In contrast with past work (Clark and Curran, 2009), which used simple transductions on category pairs, our approach uses richer transductions attached to single categories. Our conversion preserves more sentences under round-trip conversion (51.1% vs. 39.6%) and is more robust. In particular, unlike past methods, ours does not require ad-hoc rules over non-local features, and so can be easily integrated into a parser.

1 Introduction

Converting the Penn Treebank (PTB, Marcus et al., 1993) to other formalisms, such as HPSG (Miyao et al., 2004), LFG (Cahill et al., 2008), LTAG (Xia, 1999), and CCG (Hockenmaier, 2003), is a complex process that renders linguistic phenomena in formalism-specific ways. Tools for reversing these conversions are desirable for downstream parser use and parser comparison. However, reversing conversions is difficult, as corpus conversions may lose information or smooth over PTB inconsistencies.

Clark and Curran (2009) developed a CCG to PTB conversion that treats the CCG derivation as a phrase structure tree and applies hand-crafted rules to every pair of categories that combine in the derivation. Because their approach does not exploit the generalisations inherent in the CCG formalism, they must resort to ad-hoc rules over non-local features of the CCG constituents being combined (when a fixed pair of CCG categories correspond to multiple PTB structures). Even with such rules, they correctly convert only 39.6% of gold CCGbank derivations.

Our conversion assigns a set of bracket instructions to each word based on its CCG category, then follows the CCG derivation, applying and combining instructions at each combinatory step to build a phrase structure tree. This requires specific instructions for each category (not all pairs), and generic operations for each combinator. We cover all categories in the development set and correctly convert 51.1% of sentences. Unlike Clark and Curran's approach, we require no rules that consider non-local features of constituents, which enables the possibility of simple integration with a CKY-based parser.

The most common errors our approach makes involve nodes for clauses and rare spans such as QPs, NXs, and NACs. Many of these errors are inconsistencies in the original PTB annotations that are not recoverable. These issues make evaluating parser output difficult, but our method does enable an improved comparison of CCG and PTB parsers.

2 Background

There has been extensive work on converting parser output for evaluation, e.g. Lin (1998) and Briscoe et al. (2002) proposed using underlying dependencies for evaluation. There has also been work on conversion to phrase structure, from dependencies (Xia and Palmer, 2001; Xia et al., 2009) and from lexicalised formalisms, e.g. HPSG (Matsuzaki and Tsujii, 2008) and TAG (Chiang, 2000; Sarkar, 2001). Our focus is on CCG to PTB conversion (Clark and Curran, 2009).

2.1 Combinatory Categorial Grammar (CCG)

The lower half of Figure 1 shows a CCG derivation (Steedman, 2000) in which each word is assigned a *category*, and *combinatory rules* are applied to adjacent categories until only one remains. Categories

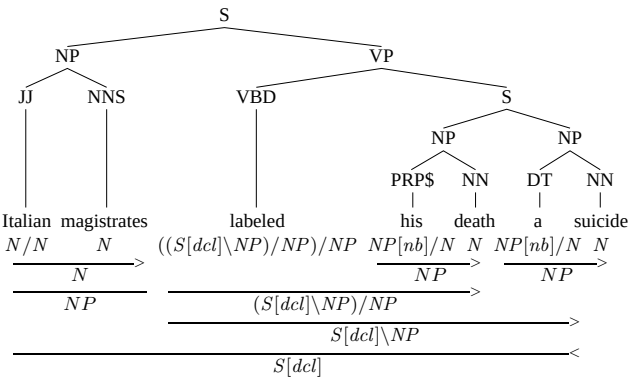

Figure 1: A crossing constituents example: *his . . . suicide* (PTB) crosses *labeled . . . death* (CCGbank).

Categories	Schema
N	create an NP
$((S[dcl]\backslash NP)/NP)/NP$	create a VP
N/N + N	place left under right
$NP[nb]/N$ + N	place left under right
$((S[dcl]\backslash NP)/NP)/NP$ + NP	place right under left
$(S[dcl]\backslash NP)/NP$ + NP	place right under left
NP + $S[dcl]\backslash NP$	place both under S

Table 1: Example C&C-CONV lexical and rule schemas.

can be atomic, e.g. the N assigned to *magistrates*, or complex functions of the form *result / arg*, where *result* and *arg* are categories and the slash indicates the argument's directionality. Combinators define how adjacent categories can combine. Figure 1 uses *function application*, where a complex category consumes an adjacent argument to form its result, e.g. $S[dcl]\backslash NP$ combines with the NP to its left to form an $S[dcl]$. More powerful combinators allow categories to combine with greater flexibility.

We cannot form a PTB tree by simply relabeling the categories in a CCG derivation because the mapping to phrase labels is many-to-many, CCG derivations contain extra brackets due to binarisation, and there are cases where the constituents in the PTB tree and the CCG derivation cross (e.g. in Figure 1).

2.2 Clark and Curran (2009)

Clark and Curran (2009), hereafter C&C-CONV, assign a *schema* to each leaf (lexical category) and rule (pair of combining categories) in the CCG derivation. The PTB tree is constructed from the CCG bottom-up, creating leaves with lexical schemas, then merging/adding sub-trees using rule schemas at each step.

The schemas for Figure 1 are shown in Table 1. These apply to create NPs over *magistrates*, *death*, and *suicide*, and a VP over *labeled*, and then combine the trees by placing one under the other at each step, and finally create an S node at the root.

C&C-CONV has sparsity problems, requiring schemas for all valid pairs of categories — at a minimum, the 2853 unique category combinations found in CCGbank. Clark and Curran (2009) create schemas for only 776 of these, handling the remainder with approximate catch-all rules.

C&C-CONV only specifies one simple schema for each rule (pair of categories). This appears reasonable at first, but frequently causes problems, e.g.:

$(N/N)/(N/N)$ + N/N
"more than" + "30" (1)
"relatively" + "small" (2)

Here either a QP bracket (1) or an ADJP bracket (2) should be created. Since both examples involve the same rule schema, C&C-CONV would incorrectly process them in the same way. To combat the most glaring errors, C&C-CONV manipulates the PTB tree with ad-hoc rules based on non-local features over the CCG nodes being combined — an approach that cannot be easily integrated into a parser.

These disadvantages are a consequence of failing to exploit the generalisations that CCG combinators define. We return to this example below to show how our approach handles both cases correctly.

3 Our Approach

Our conversion assigns a set of instructions to each lexical category and defines generic operations for each combinator that combine instructions. Figure 2 shows a typical instruction, which specifies the node to create and where to place the PTB trees associated with the two categories combining. More complex operations are shown in Table 2. Categories with multiple arguments are assigned one instruction per argument, e.g. *labeled* has three. These are applied one at a time, as each combinatory step occurs.

For the example from the previous section we begin by assigning the instructions shown in Table 3. Some of these can apply immediately as they do not involve an argument, e.g. *magistrates* has (NP f).

One of the more complex cases in the example is *Italian*, which is assigned (NP f {a}). This creates a new bracket, inserts the functor's tree, and flattens and inserts the argument's tree, producing:

(NP (JJ Italian) (NNS magistrates))

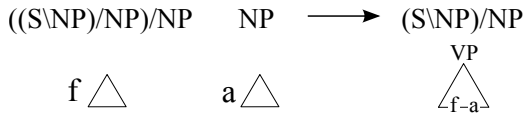

Figure 2: An example function application. Top row: CCG rule. Bottom row: applying instruction (VP f a).

Symbol	Meaning	Example
(X f a)	Add an X bracket around functor and argument	(VP f a)
{ }	Flatten enclosed node	(N f {a})
X*	Use same label as arg. or default to X	(S* f {a})
f_i	Place subtrees	(PP f_0 (S $f_{1..k}$ a))

Table 2: Types of operations in instructions.

For the complete example the final tree is almost correct but omits the S bracket around the final two NPs. To fix our example we could have modified our instructions to use the final symbol in Table 2. The subscripts indicate which subtrees to place where. However, for this particular construction the PTB annotations are inconsistent, and so we cannot recover without introducing more errors elsewhere.

For combinators other than function application, we combine the instructions in various ways. Additionally, we vary the instructions assigned based on the POS tag in 32 cases, and for the word *not*, to recover distinctions not captured by CCGbank categories alone. In 52 cases the later instructions depend on the structure of the argument being picked up. We have sixteen special cases for non-combinatory binary rules and twelve special cases for non-combinatory unary rules.

Our approach naturally handles our QP vs. ADJP example because the two cases have different lexical categories: $((N/N)/(N/N))\backslash(S[adj]\backslash NP)$ on *than* and $(N/N)/(N/N)$ on *relatively*. This lexical difference means we can assign different instructions to correctly recover the QP and ADJP nodes, whereas C&C-CONV applies the same schema in both cases as the categories combining are the same.

4 Evaluation

Using sections 00-21 of the treebanks, we hand-crafted instructions for 527 lexical categories, a process that took under 100 hours, and includes all the categories used by the C&C parser. There are 647 further categories and 35 non-combinatory binary rules in sections 00-21 that we did not annotate. For unannotated categories, we use the instructions of the result category with an added instruction.

Category	Instruction set
N	(NP f)
N/N_1	(NP f {a})
$NP[nb]/N_1$	(NP f {a})
$((S[dcl]\backslash NP_3)/NP_2)/NP_1$	(VP f a)
	(VP {f} a)
	(S a f)

Table 3: Instruction sets for the categories in Figure 1.

System	Data	P	R	F	Sent.
C&C CONV	00 (all)	95.37	93.67	94.51	39.6
	00 (len $\leq$ 40)	95.85	94.39	95.12	42.1
	23 (all)	95.33	93.95	94.64	39.7
	23 (len $\leq$ 40)	95.44	94.04	94.73	41.9
This Work	00 (all)	96.69	96.58	96.63	51.1
	00 (len $\leq$ 40)	96.98	96.77	96.87	53.6
	23 (all)	96.49	96.11	**96.30**	**51.4**
	23 (len $\leq$ 40)	96.57	96.21	96.39	53.8

Table 4: PARSEVAL Precision, Recall, F-Score, and exact sentence match for converted gold CCG derivations.

Table 4 compares our approach with C&C-CONV on gold CCG derivations. The results shown are as reported by EVALB (Abney et al., 1991) using the Collins (1997) parameters. Our approach leads to increases on all metrics of at least 1.1%, and increases exact sentence match by over 11% (both absolute).

Many of the remaining errors relate to missing and extra clause nodes and a range of rare structures, such as QPs, NACs, and NXs. The only other prominent errors are single word spans, e.g. extra or missing ADVPs. Many of these errors are unrecoverable from CCGbank, either because inconsistencies in the PTB have been smoothed over or because they are genuine but rare constructions that were lost.

4.1 Parser Comparison

When we convert the output of a CCG parser, the PTB trees that are produced will contain errors created by our conversion as well as by the parser. In this section we are interested in comparing parsers, so we need to factor out errors created by our conversion.

One way to do this is to calculate a projected score (PROJ), as the parser result over the oracle result, but this is a very rough approximation. Another way is to evaluate only on the 51% of sentences for which our conversion from gold CCG derivations is perfect (CLEAN). However, even on this set our conversion

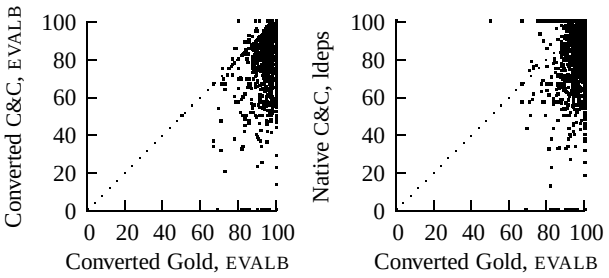

Figure 3: For each sentence in the treebank, we plot the converted parser output against gold conversion (left), and the original parser evaluation against gold conversion (right). Left: Most points lie below the diagonal, indicating that the quality of converted parser output (y) is upper bounded by the quality of conversion on gold parses (x). Right: No clear correlation is present, indicating that the set of sentences that are converted best (on the far right), are not necessarily easy to parse.

introduces errors, as the parser output may contain categories that are harder to convert.

Parser F-scores are generally higher on CLEAN, which could mean that this set is easier to parse, or it could mean that these sentences don't contain annotation inconsistencies, and so the parsers aren't incorrect for returning the true parse (as opposed to the one in the PTB). To test this distinction we look for correlation between conversion quality and parse difficulty on another metric. In particular, Figure 3 (right) shows CCG labeled dependency performance for the C&C parser vs. CCGbank conversion PARSEVAL scores. The lack of a strong correlation, and the spread on the line $x = 100$, supports the theory that these sentences are not necessarily easier to parse, but rather have fewer annotation inconsistencies.

In the left plot, the y-axis is PARSEVAL on converted C&C parser output. Conversion quality essentially bounds the performance of the parser. The few points above the diagonal are mostly short sentences on which the C&C parser uses categories that lead to one extra correct node. The main constructions on which parse errors occur, e.g. PP attachment, are rarely converted incorrectly, and so we expect the number of errors to be cumulative. Some sentences are higher in the right plot than the left because there are distinctions in CCG that are not always present in the PTB, e.g. the argument-adjunct distinction.

Table 5 presents F-scores for three PTB parsers and three CCG parsers (with their output converted by our method). One interesting comparison is between the PTB parser of Petrov and Klein (2007) and the CCG parser of Fowler and Penn (2010), which use the same underlying parser. The performance gap is partly due to structures in the PTB that are not recoverable from CCGbank, but probably also indicates that the split-merge model is less effective in CCG, which has far more symbols than the PTB.

Sentences	CLEAN	ALL	PROJ
Converted gold CCG			
CCGbank	100.0	96.3	–
Converted CCG			
Clark and Curran (2007)	90.9	85.5	88.8
Fowler and Penn (2010)	90.9	86.0	89.3
Auli and Lopez (2011)	91.7	86.2	89.5
Native PTB			
Klein and Manning (2003)	89.8	85.8	–
Petrov and Klein (2007)	93.6	90.1	–
Charniak and Johnson (2005)	94.8	91.5	–

Table 5: F-scores on section 23 for PTB parsers and CCG parsers with their output converted by our method. CLEAN is only on sentences that are converted perfectly from gold CCG (51%). ALL is over all sentences. PROJ is a projected F-score (ALL result / CCGbank ALL result).

It is difficult to make conclusive claims about the performance of the parsers. As shown earlier, CLEAN does not completely factor out the errors introduced by our conversion, as the parser output may be more difficult to convert, and the calculation of PROJ only roughly factors out the errors. However, the results do suggest that the performance of the CCG parsers is approaching that of the Petrov parser.

5 Conclusion

By exploiting the generalised combinators of the CCG formalism, we have developed a new method of converting CCG derivations into PTB-style trees. Our system, which is publicly available[1], is more effective than previous work, increasing exact sentence match by more than 11% (absolute), and can be directly integrated with a CCG parser.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful suggestions. This research was supported by a General Sir John Monash Fellowship, the Office of Naval Research under MURI Grant No. N000140911081, ARC Discovery grant DP1097291, and the Capital Markets CRC.

[1] http://code.google.com/p/berkeley-ccg2pst/

References

S. Abney, S. Flickenger, C. Gdaniec, C. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. Procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proceedings of the workshop on Speech and Natural Language*, pages 306–311.

Michael Auli and Adam Lopez. 2011. A comparison of loopy belief propagation and dual decomposition for integrated ccg supertagging and parsing. In *Proceedings of ACL*, pages 470–480.

Ted Briscoe, John Carroll, Jonathan Graham, and Ann Copestake. 2002. Relational evaluation schemes. In *Proceedings of the Beyond PARSEVAL Workshop at LREC*, pages 4–8.

Aoife Cahill, Michael Burke, Ruth O'Donovan, Stefan Riezler, Josef van Genabith, and Andy Way. 2008. Wide-coverage deep statistical parsing using automatic dependency structure annotation. *Computational Linguistics*, 34(1):81–124.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of ACL*, pages 173–180.

David Chiang. 2000. Statistical parsing with an automatically-extracted tree adjoining grammar. In *Proceedings of ACL*, pages 456–463.

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

Stephen Clark and James R. Curran. 2009. Comparing the accuracy of CCG and penn treebank parsers. In *Proceedings of ACL*, pages 53–56.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of ACL*, pages 16–23.

Timothy A. D. Fowler and Gerald Penn. 2010. Accurate context-free parsing with combinatory categorial grammar. In *Proceedings of ACL*, pages 335–344.

Julia Hockenmaier. 2003. *Data and models for statistical parsing with Combinatory Categorial Grammar*. Ph.D. thesis, School of Informatics, The University of Edinburgh.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of ACL*, pages 423–430.

Dekang Lin. 1998. A dependency-based method for evaluating broad-coverage parsers. *Natural Language Engineering*, 4(2):97–114.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2):313–330.

Takuya Matsuzaki and Jun'ichi Tsujii. 2008. Comparative parser performance analysis across grammar frameworks through automatic tree conversion using synchronous grammars. In *Proceedings of Coling*, pages 545–552.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii. 2004. Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the penn treebank. In *Proceedings of IJCNLP*, pages 684–693.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of NAACL*, pages 404–411.

Anoop Sarkar. 2001. Applying co-training methods to statistical parsing. In *Proceedings of NAACL*, pages 1–8.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press.

Fei Xia and Martha Palmer. 2001. Converting dependency structures to phrase structures. In *Proceedings of HLT*, pages 1–5.

Fei Xia, Owen Rambow, Rajesh Bhatt, Martha Palmer, and Dipti Misra Sharma. 2009. Towards a multi-representational treebank. In *Proceedings of the 7th International Workshop on Treebanks and Linguistic Theories*, pages 159–170.

Fei Xia. 1999. Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of the Natural Language Processing Pacific Rim Symposium*, pages 398–403.