Using CCG categories to improve Hindi dependency parsing

Bharat Ram Ambati **Tejaswini Deoskar** **Mark Steedman**
Institute for Language, Cognition and Computation
School of Informatics, University of Edinburgh
bharat.ambati@ed.ac.uk, {tdeoskar,steedman}@inf.ed.ac.uk

Abstract

We show that informative lexical categories from a strongly lexicalised formalism such as Combinatory Categorial Grammar (CCG) can improve dependency parsing of Hindi, a free word order language. We first describe a novel way to obtain a CCG lexicon and treebank from an existing dependency treebank, using a CCG parser. We use the output of a supertagger trained on the CCGbank as a feature for a state-of-the-art Hindi dependency parser (Malt). Our results show that using CCG categories improves the accuracy of Malt on long distance dependencies, for which it is known to have weak rates of recovery.

1 Introduction

As compared to English, many Indian languages including Hindi have a freer word order and are also morphologically richer. These characteristics pose challenges to statistical parsers. Today, the best dependency parsing accuracies for Hindi are obtained by the shift-reduce parser of Nivre et al. (2007) (Malt). It has been observed that Malt is relatively accurate at recovering short distance dependencies, like arguments of a verb, but is less accurate at recovering long distance dependencies like co-ordination, root of the sentence, etc (Mcdonald and Nivre, 2007; Ambati et al., 2010).

In this work, we show that using CCG lexical categories (Steedman, 2000), which contain subcategorization information and capture long distance dependencies elegantly, can help Malt with those dependencies. Section 2 first shows how we extract a CCG lexicon from an existing Hindi dependency treebank (Bhatt et al., 2009) and then use it to create a Hindi CCGbank. In section 3, we develop a supertagger using the CCGbank and explore different ways of providing CCG categories from the supertagger as features to Malt. Our results show that using CCG categories can help Malt by improving the recovery of long distance relations.

2 A CCG Treebank from a Dependency Treebank

There have been some efforts at automatically extracting treebanks of CCG derivations from phrase structure treebanks (Hockenmaier and Steedman, 2007; Hockenmaier, 2006; Tse and Curran, 2010), and CCG lexicons from dependency treebanks (Çakıcı, 2005). Bos et al. (2009) created a CCGbank from an Italian dependency treebank by converting dependency trees into phrase structure trees and then applying an algorithm similar to Hockenmaier and Steedman (2007). In this work, following Çakıcı (2005), we first extract a Hindi CCG lexicon from a dependency treebank. We then use a CKY parser based on the CCG formalism to automatically obtain a treebank of CCG derivations from this lexicon, a novel methodology that may be applicable to obtaining CCG treebanks in other languages as well.

2.1 Hindi Dependency Treebank

In this paper, we work with a subset of the Hindi Dependency Treebank (HDT ver-0.5) released as part of Coling 2012 Shared Task on parsing (Bharati et al., 2012). HDT is a multi-layered dependency treebank (Bhatt et al., 2009) annotated with morpho-syntactic (morphological, part-of-speech and chunk information) and syntactico-semantic (dependency) information (Bharati et al., 2006; Bharati et al., 2009). Dependency labels are fine-grained, and mark dependencies that are syntactico-semantic in nature, such as agent (usually corresponding to subject), patient (object), and time and place expressions. There are special labels to mark long distance relations like relative clauses, co-ordination etc

Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, pages 604–609,
Sofia, Bulgaria, August 4-9 2013.

(Bharati et al., 1995; Bharati et al., 2009).

The treebank contains 12,041 training, 1,233 development and 1,828 testing sentences with an average of 22 words per sentence. We used the CoNLL format[1] for our purposes, which contains word, lemma, pos-tag, and coarse pos-tag in the WORD, LEMMA, POS, and CPOS fields respectively and morphological features and chunk information in the FEATS column.

2.2 Algorithm

We first made a list of argument and adjunct dependency labels in the treebank. For e.g., dependencies with the label `k1` and `k2` (corresponding to subject and object respectively) are considered to be arguments, while labels like `k7p` and `k7t` (corresponding to place and time expressions) are considered to be adjuncts. For readability reasons, we will henceforth refer to dependency labels with their English equivalents (e.g., SUBJ, OBJ, PURPOSE, CASE for `k1`, `k2`, `rt`, `lwg__psp` respectively).

Starting from the root of the dependency tree, we traverse each node. The category of a node depends on both its parent and children. If the node is an argument of its parent, we assign the chunk tag of the node (e.g., NP, PP) as its CCG category. Otherwise, we assign it a category of `X|X`, where `X` is the parent's *result* category and | is directionality (\ or /), which depends on the position of the node w.r.t. its parent. The *result* category of a node is the category obtained once its arguments are resolved. For example, `S`, is the result category for `(S\NP)\NP`. Once we get the partial category of a node based on the node's parent information, we traverse through the children of the node. If a child is an argument, we add that child's chunk tag, with appropriate directionality, to the node's category. The algorithm is sketched in Figure 1 and an example of a CCG derivation for a simple sentence (marked with chunk tags; NP and VGF are the chunk tags for noun and finite verb chunks respectively.) is shown in Figure 2. Details of some special cases are described in the following subsections.

We created two types of lexicon. In Type 1, we keep morphological information in noun categories and in Type 2, we don't. For example, consider a noun chunk 'raam ne (Ram ERG)'. In Type 1, CCG categories for 'raam' and 'ne' are `NP` and `NP[ne]\NP` respectively. In Type 2, respective CCG categories for 'raam' and 'ne' are `NP` and `NP\NP`. Morphological information such as case (e.g., Ergative case - 'ne') in noun categories is expected to help with determining their dependency labels, but makes the lexicon more sparse.

[1] http://nextens.uvt.nl/depparse-wiki/DataFormat

```
ModifyTree(DependencyTree tree);
for (each node in tree):
    handlePostPositionMarkers(node);
    handleCoordination(node);
    handleRelativeClauses(node);
    if (node is an argument of parent):
      cat = node.chunkTag;
    else:
      prescat = parent.resultCategory;
      cat = prescat + getDir(node, parent) + prescat;
   for(each child of node):
      if (child is an argument of node):
         cat = cat + getDir(child, node) + child.chunkTag;
```

Figure 1: Algorithm for extracting CCG lexicon from a dependency tree.

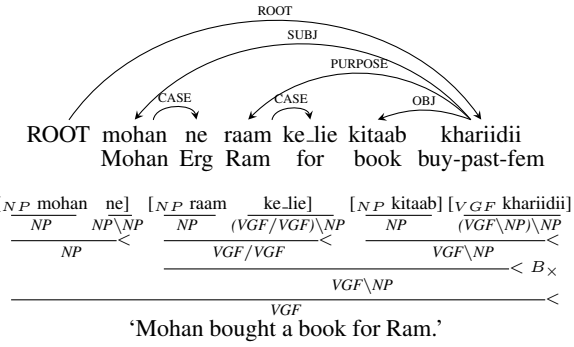


Figure 2: An example dependency tree with its CCG derivation (Erg = Ergative case).

2.3 Morphological Markers

In Hindi, morphological information is encoded in the form of post-positional markers on nouns, and tense, aspect and modality markers on verbs. A post-positional marker following a noun plays the role of a case-marker (e.g., 'raam ne (Ram ERG)', here 'ne' is the ergative case marker) and can also have a role similar to English prepositions (e.g., 'mej par (table on)'). Post-positional markers on nouns can be simple one word expressions like 'ne' or 'par' or can be multiple words as in 'raam ke lie (Ram for)'. Complex post position markers as a whole give information about how the head noun or verb behaves. We merged complex post position markers into single words like 'ke_lie' so

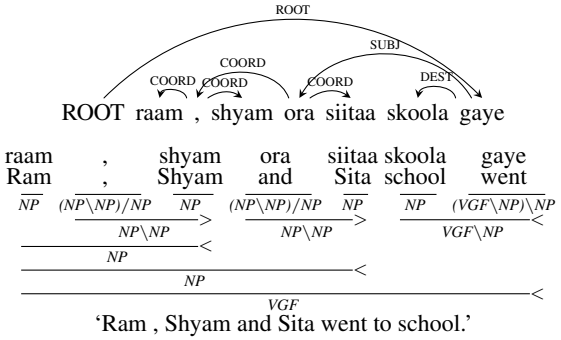
that the entire marker gets a single CCG category.

For an adjunct like 'raam ke_lie (for Ram)' in Figure 2, 'raam' can have a CCG category VGF/VGF as it is the head of the chunk and 'ke_lie' a category of (VGF/VGF)\(VGF/VGF). Alternatively, if we pass the adjunct information to the post-position marker ('ke_lie'), and use the chunk tag 'NP' as the category for the head word ('raam'), then categories of 'raam' and 'ke_lie' are NP and (VGF/VGF)\NP respectively. Though both these analysis give the same semantics, we chose the latter as it leads to a less sparse lexicon. Also, adjuncts that modify adjacent adjuncts are assigned identical categories X/X making use of CCG's composition rule and following Çakıcı (2005).

2.4 Co-ordination

The CCG category of a conjunction is (X\X)/X, where a conjunction looks for a child to its right and then a child to its left. To handle conjunction with multiple children, we modified the dependency tree, as follows.

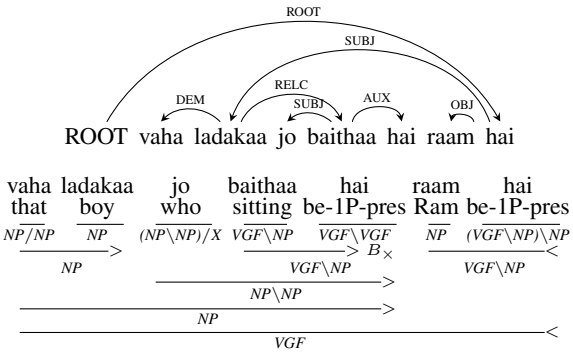
For the example given below, in the original dependency tree, the conjunction **ora** 'and' has three children 'Ram' , 'Shyam' and 'Sita'. We modified the original dependency tree and treat the comma ',' as a conjunction. As a result, ',' will have 'Ram' and 'Shyam' as children and 'and' will have ',' and 'Sita' as children. It is straightforward to convert this tree into the original dependency tree for the purpose of evaluation/comparison with other dependency parsers.



'Ram , Shyam and Sita went to school.'

2.5 Relative Clauses

In English, relative clauses have the category type NP\NP, where they combine with a noun phrase on the left to give a resulting noun phrase. Hindi, due to its freer word order, has relative clauses of the type NP\NP or NP/NP based on the position of the relative clause with respect to the head noun. Similar to English, the relative pronoun has a CCG category of (NP|NP)|X where directionality depends on the position of the relative pronoun in the clause and the category X depends on the grammatical role of the relative pronoun. In the following example, X is VGF\NP



'The boy who is sitting is Ram'

2.6 CCG Lexicon to Treebank conversion

We use a CCG parser to convert the CCG lexicon to a CCG treebank as conversion to CCG trees directly from dependency trees is not straightforward. Using the above algorithm, we get one CCG category for every word in a sentence. We then run a non-statistical CKY chart parser based on the CCG formalism[2], which gives CCG derivations based on the lexical categories. This gives multiple derivations for some sentences. We rank these derivations using two criteria. The first criterion is correct recovery of the gold dependency tree. Derivations which lead to gold dependencies are given higher weight. In the second criterion, we prefer derivations which yield intra-chunk dependencies (e.g., verb and auxiliary) prior to inter-chunk (e.g., verb and its arguments). For example, morphological markers (which lead to intra-chunk dependencies) play a crucial role in identifying correct dependencies . Resolving these dependencies first helps parsers in better identification of inter-chunk dependencies such as argument structure of the verb (Ambati, 2011). We thus extract the best derivation for each sentence and create a CCGbank for Hindi. Coverage, i.e., number of sentences for which we got at least one complete derivation, using this lexicon is 96%. The remaining 4% are either cases of wrong annotations in the original treebank, or rare constructions which are currently not handled by our conversion algorithm.

[2]http://openccg.sourceforge.net/

3 Experiments

In this section, we first describe the method of developing a supertagger using the CCGbank. We then describe different ways of providing CCG categories from the supertagger as features to a state-of-the-art Hindi Dependency parser (Malt).

We did all our experiments using both gold features (pos, chunk and morphological information) provided in the treebank and automatic features extracted using a Hindi shallow parser[3]. We report results with automatic features but we also obtained similar improvements with gold features.

3.1 Category Set

For supertagging, we first obtained a category set from the CCGbank training data. There are 2,177 and 718 category types in Type 1 (with morph. information) and Type 2 (without morph. information) data respectively. Clark and Curran (2004) showed that using a frequency cutoff can significantly reduce the size of the category set with only a small loss in coverage. We explored different cut-off values and finally used a cutoff of 10 for building the tagger. This reduced the category types to 376 and 202 for Type 1 and Type 2 respectively. The percent of category tokens in development data that don't appear in the category set entrailed by this cut-off are 1.39 & 0.47 for Type 1 and Type 2 respectively.

3.2 Supertagger

Following Clark and Curran (2004), we used a Maximum Entropy approach to build our supertagger. We explored different features in the context of a 5-word window surrounding the target word. We used features based on WORD (*w*), LEMMA (*l*), POS (*p*), CPOS (*c*) and the FEATS (*f*) columns of the CoNLL format. Table 1 shows the impact of different features on supertagger performance. Experiments 1, 2, 3 have current word (w_i) features while Experiments 4, 5, 6 show the impact of contextual and complex bi-gram features.

Accuracy of the supertagger after Experiment 6 is 82.92% and 84.40% for Type 1 and Type 2 data respectively. As the number of category types in Type 1 data (376) are much higher than in Type 2 (202), it is not surprising that the performance of the supertagger is better for Type 2 as compared to Type 1.

[3]http://ltrc.iiit.ac.in/analyzer/hindi/

Experiments: Features	*Accuracy*	
	Type 1	*Type 2*
Exp 1: w_i, p_i	75.14	78.47
Exp 2: *Exp 1* + l_i, c_i	77.58	80.17
Exp 3: *Exp 2* + f_i	80.43	81.88
Exp 4: *Exp 3* +w_{i-1},w_{i-2}, p_{i-1},p_{i-2}, w_{i+1}, w_{i+2}, p_{i+1}, p_{i+2}	82.72	84.15
Exp 5: *Exp 4* + w_ip_i, w_ic_i, w_if_i, p_if_i	82.81	84.29
Exp 6: *Exp 5* + $w_{i-2}w_{i-1}$, $w_{i-1}w_i$, w_iw_{i+1}, $w_{i+1}w_{i+2}$, $p_{i-2}p_{i-1}$, $p_{i-1}p_i$, p_ip_{i+1}, $p_{i+1}p_{i+2}$	**82.92**	**84.40**

Table 1: Impact of different features on the supertagger performance for development data.

3.3 Dependency Parsing

There has been a significant amount of work on Hindi dependency parsing in the recent past (Husain, 2009; Husain et al., 2010; Bharati et al., 2012). Out of all these efforts, state-of-the-art accuracy is achieved using the Malt parser. We first run Malt with previous best settings (Bharati et al., 2012) which use the arc-standard parsing algorithm with a liblinear learner, and treat this as our baseline. We compare and analyze results after adding supertags as features with this baseline.

3.4 Using Supertags as Features to Malt

Çakıcı (2009) showed that using gold CCG categories extracted from dependency trees as features to MST parser (McDonald et al., 2006) boosted the performance for Turkish. But using automatic categories from a supertagger radically decreased performance in their case as supertagger accuracy was very low. We have explored different ways of incorporating both gold CCG categories and supertagger-provided CCG categories into dependency parsing. Following Çakıcı (2009), instead of using supertags for all words, we used supertags which occurred at least K times in the training data, and backed off to coarse POS-tags otherwise. We experimented with different values of K and found that K=15 gave the best results.

We first provided gold CCG categories as features to the Malt parser and then provided the output of the supertagger described in section 3.2. We did all these experiments with both Type 1 and Type 2 data. Unlabelled Attachment Scores (UAS) and Labelled Attachment Scores (LAS) for Malt

are shown in Table 2. As expected, gold CCG categories boosted UAS and LAS by around 6% and 7% respectively, for both Type 1 and Type 2 data. This clearly shows that the rich subcategorization information provided by CCG categories can help a shift-reduce parser. With automatic categories from a supertagger, we also got improvements over the baseline, for both Type 1 and Type 2 data. All the improvements are statistically significant (McNemar's test, $p < 0.01$).

With gold CCG categories, Type 1 data gave slightly better improvements over Type 2 as Type 1 data has richer morphological information. But, in the case of supertagger output, Type 2 data gave more improvements over the baseline Malt as compared to Type 1. This is because the performance of the supertagger on Type 2 data is slightly better than that of Type 1 data (see Table 1).

Experiment	*Development*		*Testing*	
	UAS	*LAS*	*UAS*	*LAS*
Malt: Baseline	89.09	83.46	88.67	83.04
Malt + Type 1 Gold	95.87*	90.79*	95.27*	90.22*
Malt + Type 2 Gold	95.73*	90.70*	95.26*	90.18*
Malt + Type 1 ST	89.54*	83.68*	88.93*	83.23*
Malt + Type 2 ST	**89.90***	**83.96***	**89.04***	**83.35***

Table 2: Supertagger impact on Hindi dependency parsing (ST=Supertags). McNemar's test, * = $p <$ 0.01.

It is interesting to notice the impact of using automatic CCG categories from a supertagger on long distance dependencies. It is known that Malt is weak at long-distance relations (Mcdonald and Nivre, 2007; Ambati et al., 2010). Providing CCG categories as features improved handling of long-distance dependencies for Malt. Figure 3 shows the F-score of the impact of CCG categories on three dependency labels, which take the major share of long distance dependencies, namely, `ROOT`, `COORD`, and `RELC`, the labels for sentence root, co-ordination, and relative clause respectively. For these relations, providing CCG categories gave an increment of 1.2%, 1.4% and 1.6% respectively over the baseline.

We also found that the impact of CCG categories is higher when the span of the dependency is longer. Figure 4 shows the F-score of the impact of CCG categories on dependencies based on the distance between words. Using CCG categories

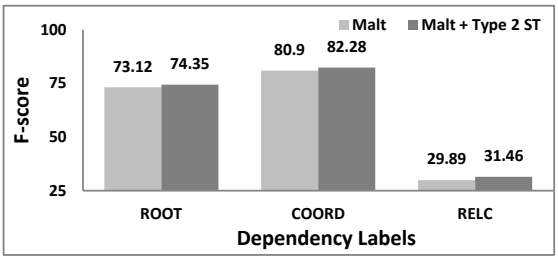


Figure 3: Label-wise impact of supertag features.

does not have much impact on short distance dependencies (1−5), which Malt is already good at. For longer range distances, 6−10, and >10, there is an improvement of 1.8% and 1.4% respectively.

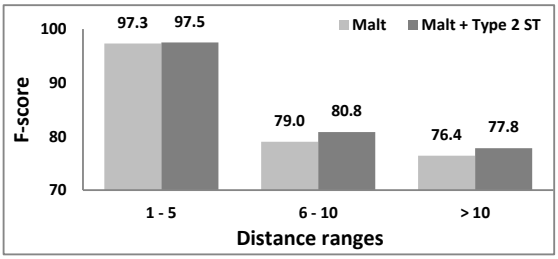


Figure 4: Impact of supertags on distance ranges.

4 Conclusion and Future Direction

We have presented an approach for automatically extracting a CCG lexicon from a dependency treebank for Hindi. We have also presented a novel way of creating a CCGbank from a dependency treebank using a CCG parser and the CCG lexicon. Unlike previous work, we obtained improvements in dependency recovery using automatic supertags, as well as gold information. We have shown that informative CCG categories improve the performance of a shift-reduce dependency parser (Malt) in recovering some long distance relations. In future work, we would like to directly train a CCG shift-reduce parser (such as Zhang and Clark (2011)'s English parser) on the Hindi CCGbank. We would also like to see the impact of generalisation of our lexicon using the free-word order formalism for CCG categories of Baldridge (2002).

Acknowledgements

We would like to thank three anonymous reviewers for their useful suggestions. This work was supported by ERC Advanced Fellowship 249520 GRAMPLUS.

References

Bharat Ram Ambati, Samar Husain, Sambhav Jain, Dipti Misra Sharma, and Rajeev Sangal. 2010. Two Methods to Incorporate 'Local Morphosyntactic' Features in Hindi Dependency Parsing. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 22–30, Los Angeles, CA, USA, June.

Bharat Ram Ambati. 2011. *Hindi Dependency Parsing and Treebank Validation*. Master's Thesis, International Institute of Information Technology - Hyderabad, India.

Jason M. Baldridge. 2002. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh, UK.

Akshar Bharati, Vineet Chaitanya, and Rajeev Sangal. 1995. Natural Language Processing: A Paninian Perspective. *Prentice-Hall of India*, pages 65–106.

Akshar Bharati, Rajeev Sangal, Dipti Misra Sharma, and Lakshmi Bai. 2006. AnnCorra: Annotating Corpora Guidelines for POS and Chunk Annotation for Indian Languages. In *Technical Report (TR-LTRC-31), LTRC, IIIT-Hyderabad*.

Akshar Bharati, Dipti Misra Sharma, Samar Husain, Lakshmi Bai, Rafiya Begum, and Rajeev Sangal. 2009. AnnCorra: TreeBanks for Indian Languages, Guidelines for Annotating Hindi TreeBank (version 2.0). http://ltrc.iiit.ac.in/MachineTrans/research/tb/DS-guidelines/DS-guidelines-ver2-28-05-09.pdf.

Akshar Bharati, Prashanth Mannem, and Dipti Misra Sharma. 2012. Hindi Parsing Shared Task. In *Proceedings of Coling Workshop on Machine Translation and Parsing in Indian Languages*, Kharagpur, India.

Rajesh Bhatt, Bhuvana Narasimhan, Martha Palmer, Owen Rambow, Dipti Misra Sharma, and Fei Xia. 2009. A multi-representational and multi-layered treebank for Hindi/Urdu. In *Proceedings of the Third Linguistic Annotation Workshop at 47th ACL and 4th IJCNLP*, pages 186–189, Suntec, Singapore.

Johan Bos, Cristina Bosco, and Alessandro Mazzei. 2009. Converting a Dependency Treebank to a Categorial Grammar Treebank for Italian. In M. Passarotti, Adam Przepiórkowski, S. Raynaud, and Frank Van Eynde, editors, *Proceedings of the Eighth International Workshop on Treebanks and Linguistic Theories (TLT8)*, pages 27–38, Milan, Italy.

Ruken Çakıcı. 2005. Automatic induction of a CCG grammar for Turkish. In *Proceedings of Student Research Workshop, 43rd Annual Meeting of the ACL*, pages 73–78.

Ruket Çakıcı. 2009. *Parser Models for a Highly Inflected Language*. Ph.D. thesis, University of Edinburgh, UK.

Stephen Clark and James R. Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of COLING-04*, pages 282–288.

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, September.

Julia Hockenmaier. 2006. Creating a CCGbank and a wide-coverage CCG lexicon for German. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 505–512, Sydney, Australia.

Samar Husain, Prashanth Mannem, Bharat Ram Ambati, and Phani Gadde. 2010. The ICON-2010 Tools Contest on Indian Language Dependency Parsing. In *Proceedings of ICON-2010 Tools Contest on Indian Language Dependency Parsing*, Kharagpur, India.

Samar Husain. 2009. Dependency Parsers for Indian Languages. In *Proceedings of the ICON09 NLP Tools Contest: Indian Language Dependency Parsing*, India.

Ryan Mcdonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and Natural Language Learning*.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 216–220, New York City, New York.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Mark Steedman. 2000. *The syntactic process*. MIT Press, Cambridge, MA, USA.

Daniel Tse and James R. Curran. 2010. Chinese CCGbank: extracting CCG derivations from the Penn Chinese Treebank. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 1083–1091, Beijing, China.

Yue Zhang and Stephen Clark. 2011. Shift-Reduce CCG Parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 683–692, Portland, Oregon, USA, June.