Towards Surface Realization with CCGs Induced from Dependencies

Michael White
Department of Linguistics
The Ohio State University
Columbus, OH 43210, USA
mwhite@ling.osu.edu

Abstract

We present a novel algorithm for inducing Combinatory Categorial Grammars from dependency treebanks, along with initial experiments showing that it can be used to achieve competitive realization results using an enhanced version of the surface realization shared task data.

1 Introduction

In the first surface realization shared task (Belz et al., 2011), no grammar-based systems achieved competitive results, as input conversion turned out to be more difficult than anticipated. Since then, Narayan & Gardent (2012) have shown that grammar-based systems can be substantially improved with error mining techniques. In this paper, inspired by recent work on converting dependency treebanks (Ambati et al., 2013) and semantic parsing (Kwiatkowksi et al., 2010; Artzi and Zettlemoyer, 2013) with Combinatory Categorial Grammar (CCG), we pursue the alternative strategy of inducing a CCG from an enhanced version of the shared task dependencies, with initial experiments showing even better results.

A silver lining of the failure of grammar-based systems in the shared task is that it revealed some problems with the data. In particular, it became evident that in cases where a constituent is annotated with multiple roles in the Penn Treebank (PTB), the partial nature of Propbank annotation and the restriction to syntactic dependency trees meant that information was lost between the surface and deep representations, leading grammar-based systems to fail for good reason. For example, Figure 1 shows that with free object relatives, only one of the two roles played by *how much manufacturing strength* is captured in the deep representation, making it difficult to linearize this phrase correctly. By contrast, Figure 2 (top)

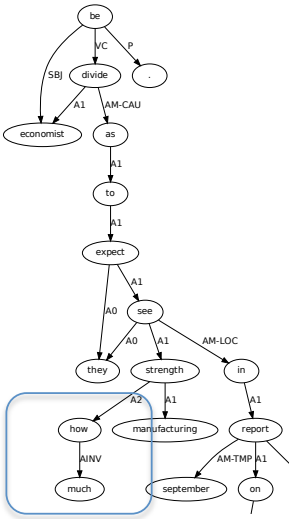


Figure 1: Shared Task Input for *Economists are divided as to [how much manufacturing strength]$_i$ they expect to see t_i in September reports on industrial production and capacity utilization , also due tomorrow* (wsj_2400.6, "deep" representation)

shows an experimental version of the shallow representation intended to capture all the syntactic dependencies in the PTB, including the additional object role played by this phrase here.[1] Including all PTB syntactic dependencies in the shallow representation makes it feasible to define a compatible CCG; at the bottom of the figure, a corresponding CCG derivation for these dependencies is shown. In the next section, we present an algorithm for inducing such derivations. In contrast to Ambati et al.'s (2013) approach, the algorithm integrates the proposal of candidate lexical categories with the derivational process, making it possible to derive categories involving unsaturated arguments, such as $\mathsf{s}_{e,\mathsf{dcl}}\backslash\mathsf{np}_x/(\mathsf{s}_{e',\mathsf{to}}\backslash\mathsf{np}_x)$; it also makes greater use of unary type-changing rules, as with Artzi & Zettlemoyer's (2013) approach.

[1] Kudos to Richard Johansson for making these enhancements available.

Unlike their approach though, it works in a broad coverage setting, and makes use of all the combinators standardly used with CCG, including ones for type-raising.

2 Inducing CCGs from Dependencies

Pseudocode for the induction algorithm is given in Figure 3. The algorithm takes as input a set of training sentences with their gold standard dependencies. We pre-processed the dependencies to make coordinating conjunctions the head, and to include features for zero-determiners. The algorithm also makes use of a seed lexicon that specifies category projection by part of speech as well as a handful of categories for function words. For example, (1) shows how a tensed verb projects to a finite clause category, while (2) shows the usual CCG category for a determiner, which here introduces a $\langle \text{NMOD} \rangle$ dependency.[2]

(1) *expect* $\vdash \mathsf{s}_{e,\mathsf{dcl}} : @_e(\textbf{expect} \wedge \langle \text{TENSE} \rangle pres)$

(2) *the* $\vdash \mathsf{np}_x/\mathsf{n}_x : @_x(\langle \text{NMOD} \rangle (d \wedge \textbf{the}))$

The algorithm begins by instantiating the lexical categories and type-changing rules that match the input dependency graph, tracking the categories in a map (*edges*) from nodes to edges (i.e., signs with a coverage vector). It then recursively visits each node in the primary dependency tree bottom up (*combineEdges*), using a local chart (*doCombos*) at each step to combine categories for adjacent phrases in all possible ways. Along the way, it creates new categories (*extendCats* and *coordCats*) and unary rules (*applyNewUnary*). For example, when processing the node for *expect* in Figure 2, the nodes for *they* and *to* are recursively processed first, deriving the categories np_{w_9} and $s_{w_{11},to} \backslash np_{w_9}/np_{w_8}$ for *they* and *to see ...*, respectively. The initial category for *expect* is then extended as shown in (3), which allows for composition with *to see ...* (as well as with a category for simple application). When there are coordination relations for a coordinating conjunction (or coordinating punctuation mark), the appropriate category for combining like types is instead constructed, as in (4). Additionally, for modifiers, unary rules are instantiated and applied, e.g. the rule for noun-noun compounds in (5).

[2] In the experiments reported here, we made use of only six (non-trivial) hand-specified categories and two type-changing rules; though we anticipate adding more initial categories to handle some currently problematic cases, the vast majority of the categories in the resulting grammar can be induced automatically.

Inputs Training set of sentences with dependencies. Initial lexicon and rules. Argument and modifier relations. Derivation scoring metric. Maximum agenda size.

Definitions *edges* is a map from dependency graph nodes to their edges, where an *edge* is a CCG sign together with a coverage bitset; *agenda* is a priority queue of edges sorted by the scoring metric; *chart* manages equivalence classes of edges; see text for descriptions of auxiliary functions such as *extendCats* and *coordCats* below.

Algorithm
bestDerivs, lexcats, unaryRules $\leftarrow \emptyset$
For each item in training set:

1. *edges*[*node*] $\leftarrow$ *instCats*(*node*), *ruleInsts*[*node*] $\leftarrow$ *instRules*(*node*), for *node* in input graph
2. *combineEdges*(*root*), with *root* of input graph
3. *bestEdge* $\leftarrow$ *unpack*(*edges*[*root*]); *bestDerivs* +$\leftarrow$ *bestEdge.sign*; *lexcats* +$\leftarrow$ *abstractedCats*(*bestEdge*), *unaryRules* +$\leftarrow$ *abstractedRules*(*bestEdge*), if *bestEdge* complete

def *combineEdges*(*node*):

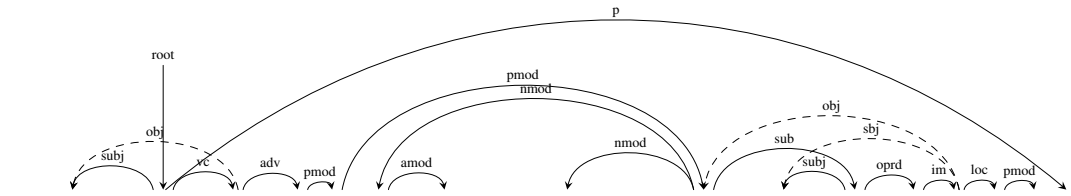
1. *combineEdges*(*child*) for *child* in *node.kids*
2. *edges*[*node*] +$\leftarrow$ *coordCats*(*node*) if *node* has coord relations, otherwise *edges*[*node*] $\leftarrow$ *extendCats*(*node*,*rels*) for argument *rels*
3. *agenda* $\leftarrow$ *edges*[*node*]; *agenda* +$\leftarrow$ *edges*[*child*] for *child* in *node.kids*; *chart* $\leftarrow \emptyset$
4. While *agenda* not empty:
   (a) *next* $\leftarrow$ *agenda.pop*
   (b) *chart* +$\leftarrow$ *next*
   (c) *doCombos*(*next*), unless *next* packed into an existing chart item
5. *edges[node]* $\leftarrow$ chart edges for *node* filtered for maximal input coverage

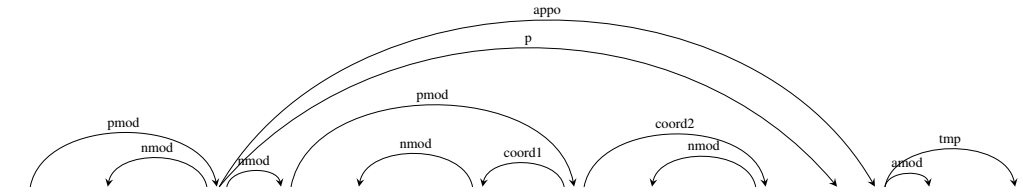
def *doCombos*(*next*):

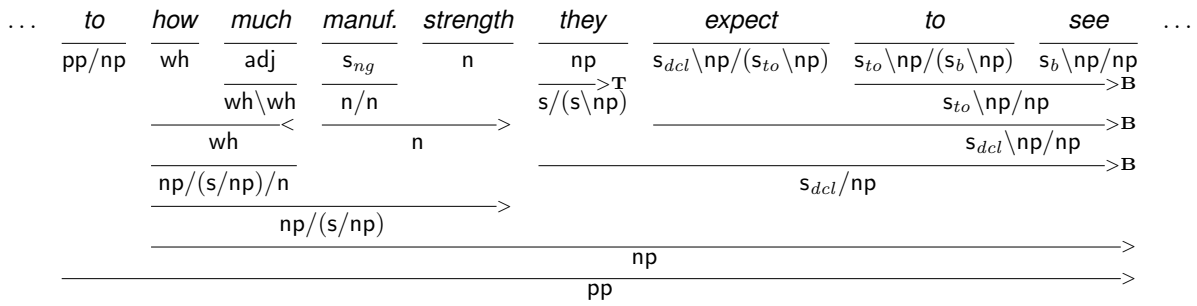
1. *agenda* +$\leftarrow$ *applyUnary*(*next*), if *next* is for *node*
2. For *item* in *chart*:
   (a) *agenda* +$\leftarrow$ *applyBinary*(*next*,*item*), if *next* is adjacent to *item*
   (b) *agenda* +$\leftarrow$ *applyNewUnary*(*next*,*item*), if *next* connected to *item* by a modifier relation

Outputs *bestDerivs, lexcats, unaryRules*

Figure 3: CCG Induction Algorithm







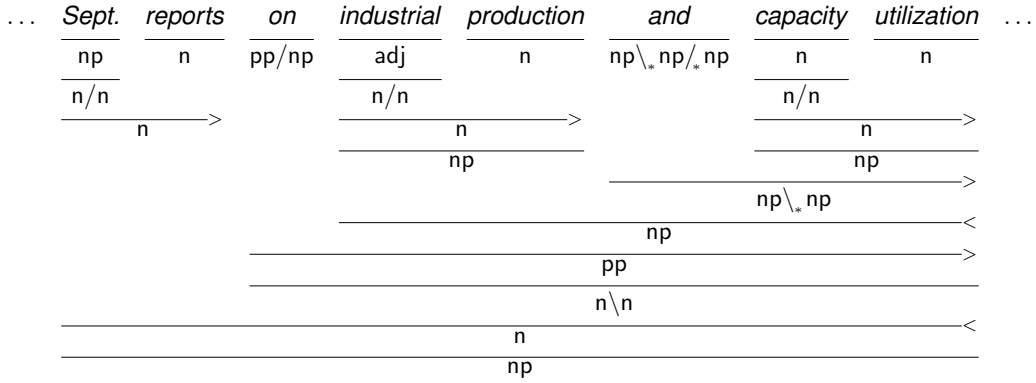


Figure 2: Augmented Syntactic Dependencies with Corresponding CCG Derivation (dashed dependencies indicate relations from additional parents beyond those in the primary tree structure)

(3) *expect* $\vdash \mathsf{s}_{w_{10},\mathsf{dcl}} \backslash \mathsf{np}_{w_9} / (\mathsf{s}_{w_{11},\mathsf{to}} \backslash \mathsf{np}_{w_9})$:
$@_{w_{10}}(\textbf{expect} \wedge \langle \text{TENSE} \rangle pres \wedge$
$\langle \text{SUBJ} \rangle w_9 \wedge \langle \text{OPRED} \rangle w_{11})$

(4) *and* $\vdash \mathsf{np}_{w_{19}} \backslash_* \mathsf{np}_{w_{18}} /_* \mathsf{np}_{w_{21}}$:
$@_{w_{19}}(\textbf{and} \wedge \langle \text{COORD1} \rangle w_{18} \wedge \langle \text{COORD2} \rangle w_{21})$

(5) $\mathsf{n}_{w_{20}} \Rightarrow \mathsf{n}_{w_{21}} / \mathsf{n}_{w_{21}}$: $@_{w_{21}}(\langle \text{NMOD} \rangle w_{20})$

At the end of the recursion, the lexical categories and type-changing rules are extracted from the highest-scoring derivation and added to the output sets, after first replacing indices such as w_{10} with variables.

3 Experiments and Future Work

We ran the induction algorithm over the standard PTB training sections (02–21), recovering complete derivations more than 90% of the time for most sections. Robust treatment of coordination, including argument cluster coordination and gapping, remains a known issue; other causes of derivation failures remain to be investigated. To select preferred derivations, we used a complexity metric that simply counts the number of steps and the number of slashes in the categories. We then trained a generative syntactic model (Hockenmaier and Steedman, 2002) and used it along with a composite language model to generate n-best realizations for reranking (White and Rajkumar, 2012), additionally using a large-scale (gigaword) language model. Development and test results appear in Table 1. Perhaps because of the expanded use of type-changing rules with simple lexical categories, the generative model and hypertagger (Espinosa et al., 2008) performed worse than expected. Combining the generative syntactic model and composite language model (GEN) with equal weight yielded a devtest BLEU score of only 0.4513, while discriminatively training the generative component models (GLOBAL) increased the score to 0.7679. Using all features increased the score to 0.8083, while doubling the beam size (ALL+) pushed the score to 0.8210, indicating that search errors may be an issue. Ablation results show that leaving out the large-scale language model (NO-BIGLM) and dependency-ordering features (NO-DEPORD) substantially drops the score.[3] Focusing only on the 80.5% of the sentences for which a complete derivation was found (COMPLETE) yielded a score of 0.8668. By comparison, realization with the native OpenCCG inputs (and the large-scale LM) on all sentences (NATIVE) yields a score more than five BLEU points higher, despite using inputs with more semantically-oriented relations and leaving out many function words, indicating that there is likely substantial room for improvement in the pre-processing and grammar induction process. Towards that end, we tried selecting the best derivations using several rounds of Viterbi EM with the generative syntactic model, but doing so did not improve realization quality.

Model	Exact	Complete	BLEU
Sect 00			
GEN	2.4	79.5	0.4513
GLOBAL	29.7	79.0	0.7679
NO-BIGLM	29.1	78.2	0.7757
NO-DEPORD	34.3	77.9	0.7956
ALL	35.8	78.4	0.8083
ALL+	36.4	80.5	**0.8210**
COMPLETE	44.4	-	0.8668
NATIVE	48.0	88.7	0.8793
Sect 23			
GEN	2.8	80.3	0.4560
GLOBAL	31.3	78.5	0.7675
ALL	37.6	77.2	0.8083
ALL+	38.1	80.4	**0.8260**
COMPLETE	47.0	-	0.8743
NATIVE	46.4	86.4	0.8694

Table 1: Development set (Section 00) & test set (Section 23) results, including exact match and complete derivation percentages and BLEU scores

A similar pattern is seen in the Section 23 results, with a competitive BLEU score of 0.8260 with the expanded beam, much higher than Narayan & Gardent's (2012) score of 0.675 with 38.8% coverage, the best previous score with a grammar-based system. This score still trails the shared task scores of the top statistical dependency realizers by several points (STUMABA-S at 0.8911 and DCU at 0.8575), though it exceeds the score of a purpose-built system using no external resources (ATT at 0.6701). In future work, we hope to close the gap with the top systems by integrating an improved ranking model into the induction process and resolving the remaining representational issues with problematic constructions.

Acknowledgments

Thanks to the anonymous reviewers, Richard Johansson and the University of Sydney Schwa Lab for helpful comments and discussion. This work was supported in part by NSF grants IIS-1143635 and IIS-1319318.

[3] All differences were statistically significant at $p < 0.01$ with paired bootstrap resampling (Koehn, 2004).

References

Bharat Ram Ambati, Tejaswini Deoskar, and Mark Steedman. 2013. Using CCG categories to improve Hindi dependency parsing. In *Proc. ACL*.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *TACL*, 1:49–62.

Anja Belz, Michael White, Dominic Espinosa, Eric Kow, Deirdre Hogan, and Amanda Stent. 2011. The first surface realisation shared task: Overview and evaluation results. In *Proc. ENLG*.

Dominic Espinosa, Michael White, and Dennis Mehay. 2008. Hypertagging: Supertagging for surface realization with CCG. In *Proc. ACL*.

Julia Hockenmaier and Mark Steedman. 2002. Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proc. ACL*.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proc. EMNLP*.

Tom Kwiatkowksi, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proc. EMNLP*.

Shashi Narayan and Claire Gardent. 2012. Error mining with suspicion trees: Seeing the forest for the trees. In *Proc. COLING*.

Michael White and Rajakrishnan Rajkumar. 2012. Minimal dependency length in realization ranking. In *Proc. EMNLP*.