Constructing Multilingual CCG Treebanks from Universal Dependencies

Anonymous ACL submission

Abstract

This paper introduces an algorithm to convert Universal Dependencies (UD) treebanks to Combinatory Categorial Grammar (CCG) treebanks. As CCG encodes almost all grammatical information into the lexicon, obtaining a high quality CCG derivation from a dependency tree is a challenging task. Our algorithm contains four main steps: binarization of dependency trees, functor/argument identification, category assignment through hand-crafted rules, and category inference for unassigned constituents. To evaluate our converted treebanks, we perform lexical, sentential, and syntactic rule coverage analysis, as well as CCG parsing experiments. We achieve over 80% conversion rate on 68 treebanks of 44 languages, and over 90% lexical coverage on 81 treebanks of 52 languages.

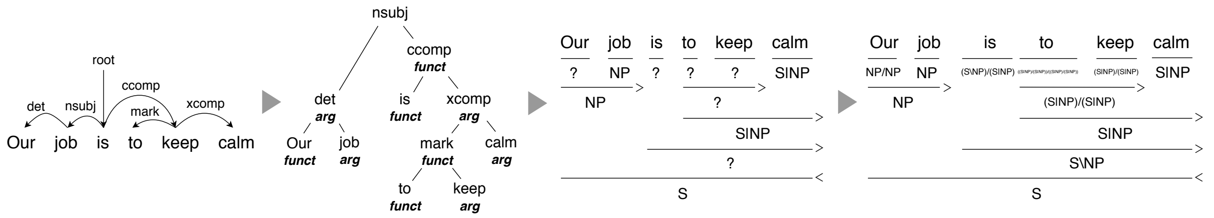
1 Introduction

Combinatory Categorial Grammar (CCG, Steedman, 2000) is a lexicalized grammar formalism that can capture both syntactic and semantic information, while allowing fast and efficient parsing. Derived syntactic structures and semantic representations can be used for various downstream tasks without task-specific training data, such as question answering (Clark et al., 2004), relation extraction (Krishnamurthy and Mitchell, 2012), and recognizing textual entailment (Martínez-Gómez et al., 2017). The English CCGbank (Hockenmaier and Steedman, 2007), one of the first available treebanks for CCG, plays an important role in the development of many wide-coverage CCG parsers for English. Having a similar resource for other languages and domains accelerates NLP research, in particular on resource-scarce languages/domains where one cannot rely on massive training data needed for training large neural network models (Peters et al., 2018; Devlin et al., 2019). Multilingual CCG resources also contribute to cross-linguistic research on syntactic/semantic theories and multilingual CCG parsing.

Since manual annotation is expensive, conversion from a source treebank is a preferable approach. Besides English, independent works have been done in the past to create CCG treebanks for several languages from source treebanks of different grammar formalisms, such as for German (Hockenmaier, 2006), Italian (Bos et al., 2009), Chinese (Tse and Curran, 2010), Japanese (Uematsu et al., 2013), and Hindi (Ambati et al., 2018). Such works often involve conversion rules that are specific to the languages and treebanks being converted, making the process difficult to adapt and generalize to other languages.

In this paper, we propose a method to create a multilingual collection of CCG treebanks by converting from dependency treebanks. To minimize the need for language-specific conversion rules, we select the Universal Dependencies (UD, Nivre et al., 2016) as our source treebanks. The UD, as of v2.8, contains over 200 treebanks in 114 languages that follow cross-linguistically consistent annotation guidelines.[1] Our goal is to develop a universal set of hand-crafted rules that can be applied to a wide range of languages in the UD, while sacrificing as little as possible the conversion quality and coverage of each converted treebank. Converted CCG treebanks can be used directly to train multilingual CCG parsers as we demonstrate in the experiments, while one can also use our resource as a starting point to further improve the quality of each treebank by adding language-specific conversion rules. Our work thus opens up a new research direction to the development of CCG resources, parsers, and semantic analysis that uses them. To obtain a CCG parser for a specific language or a domain, one only needs to develop a dependency treebank based on UD, possibly with additional language-specific conversion rules.

[1] https://universaldependencies.org/.



(1-2) Binarization & functor/argument identification

(3) Category assignment

(4) Category inference for unassigned categories

Figure 1: Example of our complete conversion process for an English sentence. The default slash direction is "|". Most slash directions ("/" or "\") can be inferred through relative positions between functors and arguments. Any undecided slash directions left at the end of the conversion process are decided via majority voting (Section 3.4).

A high-level overview of our conversion process is illustrated in Figure 1. Since CCG derivations are binary in nature, we first binarize dependency trees based on a pre-defined obliqueness hierarchy. Subsequently, for each relation in the dependency trees, we apply a hand-crafted rule that assigns CCG categories to associated constituents. To take into account the varied word-order tendencies of different languages, we use a neutral slash direction "|" when designing our rules. Finally, we infer the categories of any unassigned categories in a top-down, recursive manner, following CCG's combinatory rules. Section 3 discusses each of the above steps in more detail.

We evaluate the effectiveness of our algorithm by performing coverage analysis and parsing experiments on the converted treebanks. Analysis results on a subset of 22 treebanks of 22 languages, as well as discussions on the strengths and limitations of our algorithm, are presented in Section 4. We include our implementation and detailed experiment results in the supplementary materials.

2 Background

2.1 Combinatory Categorial Grammar

CCG is a strongly lexicalized grammar formalism, in which words are assigned syntactic *categories* that govern how they interact with other constituents. There are two types of categories: *atomic categories*, such as S and NP, and *complex categories*, which are usually in the form of X/Y or $X\backslash Y$, with X and Y being categories themselves. X/Y (or $X\backslash Y$) takes an argument Y to the right (or left), and yields a result X.

CCG also contains a set of rules that defines how categories can combine with each other. Table 1 shows a list of basic combinatory rules used in CCG. In addition, non-combinatory rules such as unary and binary type-changing rules are often included (e.g. $S\backslash NP \Rightarrow NP\backslash NP$), as they have been shown to alleviate the problem of category proliferation during treebank conversion (Hockenmaier and Steedman, 2002).

Forward Application (>)	X/Y	Y	⇒	X
Backward Application (<)	Y	X\Y	⇒	X
Forward Composition (>**B**)	X/Y	Y/Z	⇒	X/Z
Backward Composition (<**B**)	Y\Z	X\Y	⇒	X\Z
Forward Crossed Composition (>$\mathbf{B}_\times$)	X/Y	Y\Z	⇒	X\Z
Backward Crossed Composition (<$\mathbf{B}_\times$)	Y/Z	X\Y	⇒	X/Z
Forward Type-raising (>**T**)	X		⇒	T/(T\X)
Backward Type-raising (<**T**)	X		⇒	T\(T/X)

Table 1: Basic CCG combinatory rules.

2.2 Universal Dependencies

UD is a project to create cross-linguistically consistent dependency annotation guidelines. As of v2.8, there are 202 treebanks in 114 languages. One main difference between UD and other dependency grammars is its treatment of function words. To achieve better parallelism among annotations of different languages, function words are treated as dependents of content words (Nivre et al., 2016). UD is being actively developed, with adjustments to dependency definitions and new features such as Enhanced Dependencies (Nivre et al., 2020). The current version of UD consists of 37 universal dependency relations, 17 universal part-of-speech (POS) tags, and 24 universal features.

2.3 Related Work

The English CCGbank (Hockenmaier and Steedman, 2007) is one of the pioneering works to create a treebank for CCG, by converting from the Penn Treebank (Marcus et al., 1993). From then

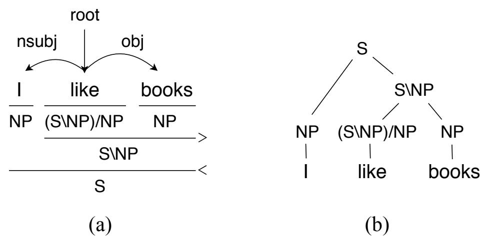


Figure 2: (a) is a standard CCG representation. (b) is an equivalent constituent structure.

on, there have been works to create CCG treebanks for German (Hockenmaier, 2006), Italian (Bos et al., 2009), Chinese (Tse and Curran, 2010), Japanese (Uematsu et al., 2013), and Hindi (Ambati et al., 2018). For works that involve converting from a dependency treebank, a common approach is to first convert to constituency trees, binarize the constituency trees, then apply conversion rules to the binarized trees. Due to a large number of cross-serial dependencies in the Hindi dependency treebank, Ambati et al. (2018) diverge from this approach by first extracting a CCG lexicon from the dependency treebank, then use a non-statistical CCG parser to attain CCG derivations. In general, all previous works involve conversion methods that are specific to the languages and treebanks being converted, making it difficult to generalize to others. Moreover, source treebanks for German, Italian, and Japanese also contain additional information regarding phrase structures (German), or predicate-argument structures (Italian, Japanese), which help alleviate certain ambiguities, such as argument-adjunct distinction. This distinction, or lack thereof, is a big obstacle when converting UD treebanks to CCG derivations.

Recently, Yoshikawa et al. (2019) propose a neural network-based model to automatically convert dependency trees to CCG derivations for parser domain adaptation. However, their method requires an existing CCG parser for fine-tuning, which is not available for most languages in UD. Evang and Bos (2016) propose an annotation projection approach to induce CCG via parallel corpora; however, the relatively small number of parallel corpora available compared to UD makes its range of applicability limited. Reddy et al. (2017) introduce an interface that converts UD dependency trees to logical forms. Compared to their work, our conversion to CCG allows more flexibility in the types of semantic representations that could be derived, such as first-order logic neo-Davidsonian representations (Bos et al., 2004), or higher-order logic representations (Mineshima et al., 2015), while also retains the syntactic information encoded in UD. Moreover, we perform larger-scale experiments and analysis on 22 languages. Our binarization method takes inspiration from their work.

3 The Conversion Process

A simple, typical CCG derivation is illustrated in Figure 2. To obtain a unique and complete derivation from a dependency tree, we need to:

1. Identify constituents.
2. Identify functors and arguments.
3. Identify the category of each constituent.

The constituent structure of a CCG derivation can be represented by a binary tree (Figure 2(b)). Since dependency trees are structurally different, a binarization step is required. As the binarized trees also represent the constituent structures of the sentences being converted, thus answering requirement (1), an *obliqueness hierarchy* is necessary to impose a correct traversal order during binarization of the dependency trees. The details of this step are explained in Section 3.1.

Identifying functors and arguments is useful in case we know the result of a CCG combination but missing one of two component categories. However, the head-dependent relations between tokens in dependency trees do not directly translate to functor-argument relations between constituents in CCG derivations. To meet requirement (2), we apply a set of rules to the binarized trees that assign a functor/argument role to each node based on its associated dependency label and the relationship with its sibling. We describe these rules and how we apply them in Section 3.2.

Finally, we fill in the category of each constituent defined in the previous steps. Requirement (3) is done in two stages: category assignment by handcrafted rules (Section 3.3), and category inference for any unassigned categories (Section 3.4).

Preprocessing: We ignore most dependency subtypes, such as `obl:tmod`, as these labels are not used consistently across treebanks of different languages. We also remove quotation marks from dependency trees, following Hockenmaier and Steedman (2007), and ignore empty nodes, which are indexed with decimal numbers in UD.

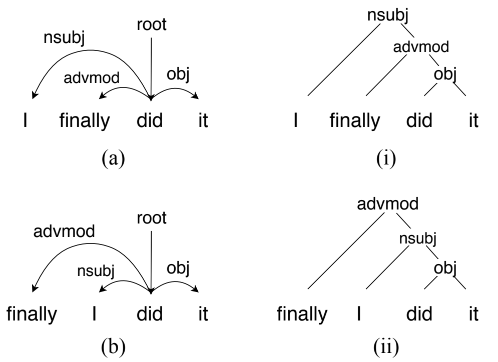


Figure 3: (a) and (b) show two sentences with a slight difference in word order. Without position information, both (a) and (b) would be binarized into (i) according to the obliqueness hierarchy (`obj` > `advmod` > `nsubj`). However, (i) leads to an invalid combination for (b), as "finally" cannot combine with "did it" due to being nonadjacent. (ii) shows the correct binarization for (b) when the condition for words' positions is applied, as it puts `nsubj` before `advmod` in the traversal order.

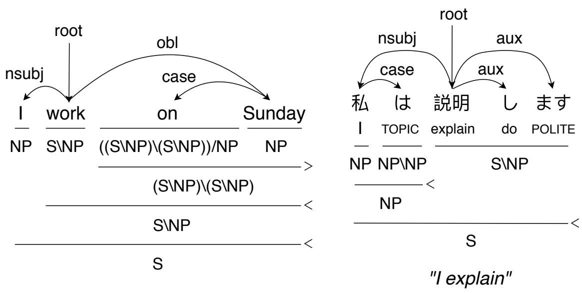


Figure 4: Examples of situations where categories for case markers may differ. On the left, case marker "on" has category of the form $X|Y$, while on the right, case marker "は" (topic marker) has category of the form $X|X$ to preserve the category of its head "私" ("I").

3.1 Binarization

We binarize dependency trees using a modified version of the binarization method proposed by Reddy et al. (2017). The method traverses the dependency trees recursively from top to bottom, and builds binarized trees by gradually adding subtrees in the order it traverses. Since a binarized tree decides which constituents combine with each other, their method depends on an obliqueness hierarchy to traverse in an order that can lead to syntactically sound combinations. However, the original method is designed to extract logical forms, and thus does not take into account the position of each constituent in a sentence. This can lead to invalid CCG combinations, as combinatory rules in CCG are only applied to string-adjacent entities.

We adapt Reddy et al.'s (2017) method to our task by adding a position-based condition: (1) for dependents of the same distance to the head, traverse in the order of the obliqueness hierarchy; (2) for dependents of different distances to the head, traverse closer dependents first. Here, "distance" is measured by the number of siblings between a dependent and its head (Figure 3).

3.2 Identifying functors/arguments

We use the binarized trees as skeletons to apply category assignment rules and category inference logics in later steps. To make category inference possible, we need to identify how constituents should be combined, and thus identify the functor/argument role of each constituent.

Our rules for identifying functors and arguments are designed around the relations between heads, arguments, and modifiers. Specifically:

1. We set the head of a head-argument relation (`nsubj`, `csubj`, `obj`, `iobj`, `xcomp`, `ccomp`, `expl`) as a functor, and its dependent as an argument.

2. We set the head of a head-modifier relation (the rest of the UD relations, with the exception of `conj`, `cc`, and `punct`) as an argument, and its dependent as a functor.

In general, the functor category in case (1) has the form $X|Y$, where X and Y are usually different categories. This means that it takes one category as input and outputs a different category. Transitive verbs ($(S\backslash NP)/NP$) is one example.

In case (2), the functor category usually has the form $X|X$, meaning it inputs and outputs the same category. Nominal modifiers or multi-word expressions ($NP|NP$) are typical cases. This rule is also helpful in the later category inference step. Given a CCG combination with the same result and argument category, we can easily infer the functor category. One exception to rule (2) is case markers (`case`). A case marker can have the form $X|Y$ if its head is a modifier to another constituent, and the form $X|X$ if its head is an argument to another constituent (Figure 4). Figure 1 shows an example of our functor/argument category assignment rules applied to a binarized tree.

`conj`, `cc`, and `punct` are special cases that do not belong to either of these rules. They follow separate non-combinatory rules for punctuations and coordinations, similar to the design of the English CCGbank (Hockenmaier and Steedman, 2007).

3.3 Category Assignment

This section describes our hand-crafted rules for category assignment. Similar to previous works on CCG induction (Bisk and Hockenmaier, 2012), we assume two atomic categories S and NP for our target grammar. Categories are assigned to internal nodes of the binarized trees obtained in the previous steps. Due to the varied word-order tendencies of different languages, we set the default slash direction of complex categories to "|", which can either take value "/" or "\". This value is either decided through heuristic rules based on relative positions of functors and arguments, or through majority voting at the end of the conversion process. The rules discussed in this section do not depend on one another, and can be applied in any order.

Root: We determine the category of a whole sentence through the `root` of the dependency tree. A sentence is assigned category NP if:

- The `root` has one of the following UPOS tags: `NOUN`, `NUM`, `PRON`, `PROPN`, `SYM`,
- The `root` does not have any nominal subject, clausal subject, or expletive children.

The sentence is assigned category $S|NP$ if:

- The `root` does not have one of the following POS tags: `NOUN`, `NUM`, `PRON`, `PROPN`, `SYM`,
- The `root` does not have any nominal subject, clausal subject, or expletive children.

Otherwise, the sentence is assigned category `S`.

Punctuations: We follow Hockenmaier and Steedman (2007) and set the category of each punctuation to be the punctuation mark itself.

Exceptions include dashes, parentheses, and variants of open and closing brackets in different languages (e.g., " 【】 " in Japanese, " 《》 " in Japanese, Chinese, and Korean). These punctuations are treated like normal constituents and carry standard CCG categories.

Adnominal clause: An adnominal clause (`acl`) modifies a nominal, and thus generally has category $NP|NP$. If an adnominal clause is not marked by any markers (`mark`), we apply a type-changing rule to change its original category to $NP|NP$ (Figure 5). The original category of an adnominal clause excluding markers is set to S if it has a clausal or a nominal subject, and $S|NP$ otherwise.

Relative clause: A relative clause is tagged as a subtype of an adjectival clause in UD

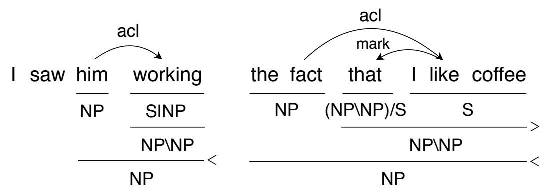


Figure 5: On the left is an example of a unary type-changing rule for `acl`. The slash direction of $NP|NP$ is by default "|", but can be inferred to be "\" based on the adjective clause's relative position to its head. On the right is an example of an adjectival clause with a marker "that", which absorbs category S of "I like coffee" and changes it to $NP|NP$.

(`acl:relcl`), but it requires a separate rule to produce a correct CCG derivation:

- The relative pronoun (identified through feature `PronType=Rel`) is assigned category $(NP|NP)|(S|NP)$, as it takes a sentence missing a subject or an object as an argument, and yields a nominal modifier.
- If a relative clause does not have a relative pronoun, its original category is set to $(S|NP)$, and is type-changed to $(NP|NP)$.
- In case of an interrogative pronoun, the constituent consisting of the interrogative pronoun and its head is assigned category $(NP|NP)|(S|NP)$.

Adverbial clause: Similarly, an adverbial clause `advcl` usually has category $(S|NP)|(S|NP)$, as it modifies a verb or a predicate. If an adverbial clause does not have any markers (`mark`), we apply a type-changing rule to change its original category to $(S|NP)|(S|NP)$. We set the original category of an adverbial clause excluding markers to S if it has a clausal or a nominal subject, and $S|NP$ otherwise. An adverbial clause can also appear in sentential modifier locations, in which case its category would be $S|S$.

Clausal complement: We assign category S to a clausal complement (`ccomp`) if it has a subject, and category $S|NP$ otherwise. An open clausal complement (`xcomp`) is assigned category `NP` if its head element has one of the following UPOS tags: `NOUN`, `NUM`, `PRON`, `PROPN`, `SYM`. Otherwise, it is also assigned category $S|NP$ (Figure 6).

Clausal subject: We only apply rules for a clausal subject (`csubj`) if it has another subject within. In this case, if a clausal subject is marked

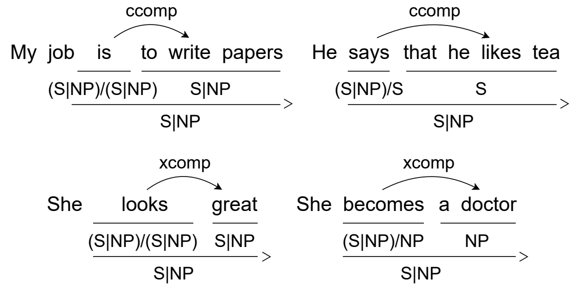


Figure 6: Examples of our rules for `ccomp/xcomp`.

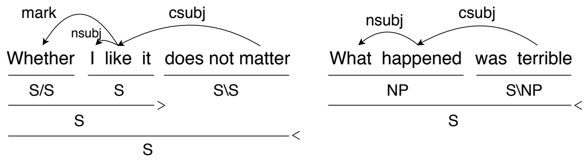


Figure 7: Examples of our rule applied to `csubj`.

by a marker (`mark`), it is assigned category S. Otherwise, it is assigned category NP (Figure 7). In other cases, clausal subjects are treated like normal core arguments, and their categories are inferred through the category inference step.

Parataxis: The UD guidelines detail five different constructions where parataxis can appear: side-by-side sentences, reported speech, news article bylines, interjected clauses, and tag questions. We treat the dependent constituent in these constructions as a modifier to its head.

Noun phrase: Category NP is assigned to tokens that have one of the UPOS tags: `NOUN`, `NUM`, `PRON`, `PROPN`, `SYM`, or non-noun tokens with accompanying determiners that act as nominal subjects or objects, *if* they do not modify any other constituents. Otherwise, their categories are inferred through the category inference step.

Vocative/dislocated/discourse/overridden disfluency elements: Since these elements are optional to the grammar and meaning of a sentence, we treat them as modifiers to their head. As a result, they carry category $X|X$, where X is the category of their head.

3.4 Category Inference

Our rules described in Section 3.3 assign categories to only a subset of constituents. As a result, there are bound to be unassigned categories. In these situations, we follow CCG's forward and backward application rules to infer the missing categories from existing ones. The category inference step is run top-down, and is repeated until no more categories can be inferred. There are two situations where additional logics are required for inference:

Punctuation: As mentioned in Section 3.3, dashes, parentheses, and other brackets follow the same CCG combinatory rules as normal constituents. Other punctuations follow a separate rule (e.g. $,\ X \Rightarrow X$), similar to the English CCGbank.

Coordination: We use the following non-combinatory rules for coordination, also similar to the English CCGbank:

$$
\begin{aligned}
conj \quad X &\Rightarrow X[conj] \\
, \quad X &\Rightarrow X[conj] \\
X[conj] \quad X &\Rightarrow X
\end{aligned}
$$

Majority voting for slash direction: Throughout the conversion process, slash direction in each category is determined through relative positions between functors and arguments. However, it is not guaranteed that all cases are covered, as shown in the example of "calm" and "to keep calm" in Figure 1. To handle these situations, we apply majority voting based on corresponding dependency relations in the binarized tree for non-terminal nodes, and on UPOS tags for terminal nodes. In the case of "to keep calm", which has a corresponding relation `ccomp`, votes are aggregated from other occurrences of `ccomp` in the whole treebank to decide a more popular slash direction. Likewise, for "calm", we collect votes from nodes with `ADJ` UPOS tag.

Since rules are applied independently, we also add a validation step to ensure the integrity of converted CCG derivations. In principle, categories inferred from applying CCG combinatory rules take priority over categories assigned by hand-crafted rules. If conflicts are found, categories inferred from applying CCG combinatory rules will override the conflicting categories.

Unprocessed dependency trees: Dependency trees with crossing arcs present a challenge for binarization. Certain treebanks, such as Ancient Greek and Latin treebanks, have a high number of sentences with crossing dependencies, which lead to significantly lower conversion rates. These sentences are currently not being converted by our algorithm, and will be the focus of our future work.

4 Evaluation

For the following experiments, treebanks with their surface stripped off, or with more than 20% of their

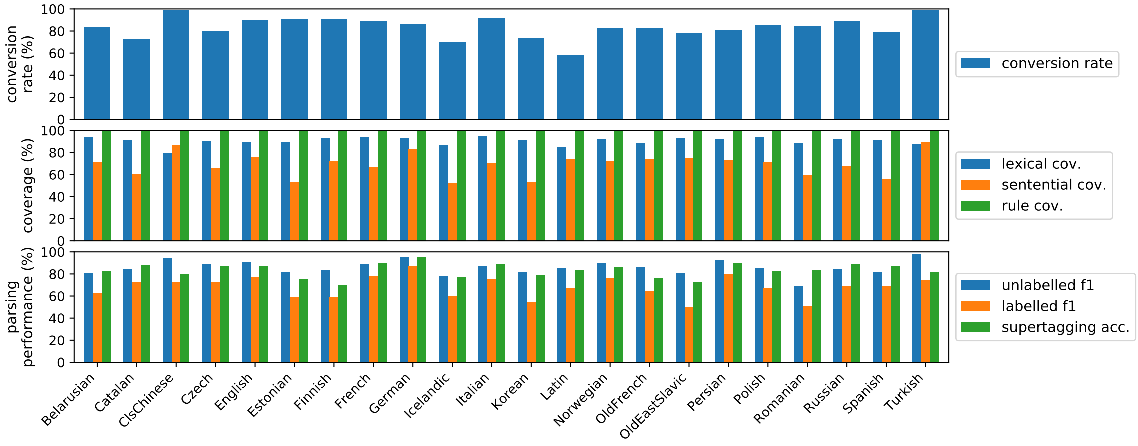


Figure 8: Conversion statistics and CCG parsing results on 22 treebanks of 22 languages, sorted by alphabetical order. Detailed numbers are reported in Table 5 of the Appendix.

sentences containing dependency `dep` or UPOS tag `X`, are excluded, as we depend on the surface for our punctuation rules, and treebanks having too many `dep` or `X` suggest an underlying problem with their annotation quality[2]. In addition, we also exclude treebanks without a proper `train/test` split, as it is necessary for our evaluation. To assess the conversion quality, we conduct lexical, sentential, and syntactic rule coverage analyses on the converted treebanks, which are commonly used metrics for evaluating induced grammar (Hockenmaier and Steedman, 2007; Tse and Curran, 2010; Uematsu et al., 2013). CCG parsing experiments are also performed on treebanks with more than 10,000 complete derivations in the training set. For languages that have more than one such treebank, we choose the largest treebank available. Figure 8 summarizes our conversion and parsing results on 22 treebanks of 22 languages. Complete conversion statistics on 105 treebanks of 65 languages tested are reported in Table 4 of the Appendix.

4.1 Conversion rate and coverage

Conversion rate: A conversion rate of a treebank measures the percentage of its sentences that are fully converted to CCG derivations. We observe better than 80% conversion rates for 68 treebanks (out of 105) of 44 languages (out of 65).

Most conversion errors can be attributed to cross-serial dependencies, dependency relation `dep`, and UPOS tag `X`. The abundance of `dep` and `X` suggests lower annotation quality of some treebanks in UD, but it also means that conversion rates can further increase by improving the source treebanks.

Lexical coverage: We treat the converted `train` set of each treebank as the gold standard, and the `dev` and `test` sets as unseen data. Lexical coverage measures how well the gold lexicon covers the categories in unseen data. Standard treatment of rare words is applied; tokens that appear less than five times are replaced by "UNK". Unassigned categories are not included in the analysis. We achieve over 90% lexical coverage on 81 treebanks of 52 languages (Table 4, Appendix).

Sentential coverage: Sentential coverage measures the percentage of sentences in unseen data that can be fully assigned with categories from the gold lexicon. We use fully converted sentences in the `dev` and `test` sets for sentential coverage analysis. The majority of our converted treebanks achieve between 55% and 70% coverage. In reality, we observe that most sentences in the `dev` and `test` sets contain only a small number of tokens not covered by the gold lexicon. This explains the high lexical coverage and average sentential coverage, and also suggests that sentential coverage can greatly benefit from minor manual correction.

Syntactic rule coverage: Syntactic rule coverage on unseen data is measured by calculating the percentage of CCG rule instantiations in `dev` and `test` sets that exist in the `train` set. We are able to achieve near-perfect coverage for all languages.

Parsing performance: We use an off-the-shelf CCG parser *depccg* (Yoshikawa et al., 2017) on 22 treebanks with more than 10,000 sentences in

[2] In UD, `dep` and `X` are only used when it is impossible to assign a more precise label, or when there are problems with the conversion/parsing software.

Frequency	Rule
30577	NP → NP/NP NP
13201	NP → NP NP\NP
13078	S\NP → (S\NP)/(S\NP) S\NP
10905	S → NP S\NP
10262	S\NP → S\NP (S\NP)\(S\NP)
8369	S\NP → (S\NP)/NP NP
6460	S → S .
5569	(S\NP)\(S\NP) → ((S\NP)\(S\NP))/NP NP
5330	NP\NP → (NP\NP)/NP NP
3767	S → S/S S

Table 2: Most frequent rule instantiations in the training set of converted English-EWT treebank.

UPOS	Category	Pct.	UPOS	Category	Pct.
VERB	(S\NP)/(S\NP)	0.301	ADP	(NP\NP)/NP	0.387
	(S\NP)/NP	0.293		((S\NP)\(S\NP))/NP	0.240
	S\NP	0.086		(S\NP)/(S\NP)	0.092
NOUN	NP	0.752	ADV	(S\NP)/(S\NP)	0.227
	NP/NP	0.095		((S\NP)\(S\NP))/NP	0.109
	NP\NP	0.023		NP/NP	0.103
ADJ	NP/NP	0.583	DET	NP/NP	0.947
	S\NP	0.134		(NP\NP)/(NP\NP)	0.014
	NP	0.059		(S\NP)/(S\NP)	0.011

Table 3: Most common categories for each UPOS tag in the training set of converted English-EWT treebank.

the training set. We run the training script for 20 epochs on each treebank, keeping all other default hyper-parameter settings. No pre-trained language model is used. Parsing performance is evaluated on the `test` split of each treebank. While the standard evaluation metric for CCG parsing is in terms of predicate-argument structure recovery, such information is not trivial to obtain from UD. As a result, we choose a more traditional metric, PARSEVAL (Black et al., 1991). With over 80% unlabelled PARSEVAL F1 and supertagging accuracy on almost all tested treebanks, our experiments show the viability of obtaining a good CCG parser for many languages from the converted treebanks.

4.2 Quality of obtained treebanks

To ensure the validity of our converted derivations, we automatically check for rule application errors at the end of the conversion algorithm. We also randomly sample and manually check 100 sentences from each `dev` set of the obtained English-EWT, Japanese-GSD, and Vietnamese-VTB treebanks:

- For English, we find 7 cases of incorrect binarization of coordination structures, one case of an incorrect category assigned to a transitive verb in a relative clause, and one case of an incorrect category assigned to a clausal subject. A side-effect of using UD is the lack of phrasal information, leading to ambiguous constituency structures in some cases.
- For Japanese, we find 48 cases of categories having incorrect slash directions, and one case of an incorrect category assigned to a noun phrase. Since Japanese sentences often lack an explicit subject, many $S|NP$ categories remain by the end of the conversion process, and are subsequently majority-voted into S/NP. As Japanese sentences are dominantly verb-final, this error can easily be handled by applying a language-specific rule that sets "\" as the default slash direction.
- For Vietnamese, we find 7 cases of incorrect binarization of coordination structures (similar to English), 8 cases of incorrect categories assigned due to annotation errors, and 4 cases of incorrect categories assigned due to errors in conversion rules.

In general, our conversion method benefits from additional language-specific rules and minor manual correction. The quality of the converted CCG treebanks is also tied to the quality of the source treebanks, as shown in the case of Vietnamese.

Similar to Bisk and Hockenmaier (2012), we also compare our obtained English CCG treebank to the English CCGbank, and observe that our induced grammar and lexicon match what we generally expect for English, with the most common rules showing high similarity to those of the English CCGbank (Table 2 and 3).

Besides the limitations listed in Section 3.4, the lack of composition rules also leads to a possible proliferation of complex categories in our derivations. For example, in Japanese and Korean treebanks, the categories of auxiliary words can be set to simply $S\backslash S$ in many cases (Lee, 2000), which can then be combined with their heads via backward composition. This also suggests how language-specific rules can improve our algorithm.

5 Conclusion

We introduced an rule-based algorithm to create CCG treebanks from UD. We believe the CCG derivations obtained from our algorithm can serve as a good starting point for CCG treebank development and CCG parsing research in many languages, from which further improvement can be made by applying additional language-specific rules or manual fine-tuning to the converted treebanks.

References

Bharat Ram Ambati, Tejaswini Deoskar, and Mark Steedman. 2018. Hindi CCGbank: CCG Treebank from the Hindi Dependency Treebank. *Language Resources and Evaluation*, 52:67–100.

Yonatan Bisk and Julia Hockenmaier. 2012. Simple robust grammar induction with combinatory categorial grammars. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

E. Black, S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19-22, 1991*.

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

Johan Bos, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. 2004. Wide-coverage semantic representations from a CCG parser. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 1240–1246, Geneva, Switzerland. COLING.

Stephen Clark, Mark Steedman, and James R. Curran. 2004. Object-extraction and question-parsing using CCG. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 111–118, Barcelona, Spain. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Kilian Evang and Johan Bos. 2016. Cross-lingual learning of an open-domain semantic parser. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 579–588, Osaka, Japan. The COLING 2016 Organizing Committee.

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 44th Annual Meeting of the Association for Computational Linguistics*, pages 505–512, Sydney, Australia. Association for Computational Linguistics.

Julia Hockenmaier and Mark Steedman. 2002. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*, Las Palmas, Canary Islands - Spain. European Language Resources Association (ELRA).

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.

Jayant Krishnamurthy and Tom Mitchell. 2012. Weakly supervised training of semantic parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 754–765, Jeju Island, Korea. Association for Computational Linguistics.

Kihwang Lee. 2000. A CCG fragment of Korean. In *Proceedings of the 14th Pacific Asia Conference on Language, Information and Computation*, pages 219–230, Waseda University International Conference Center, Tokyo, Japan. PACLIC 14 Organizing Committee.

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

Pascual Martínez-Gómez, Koji Mineshima, Yusuke Miyao, and Daisuke Bekki. 2017. On-demand injection of lexical knowledge for recognising textual entailment. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 710–720, Valencia, Spain. Association for Computational Linguistics.

Koji Mineshima, Pascual Martínez-Gómez, Yusuke Miyao, and Daisuke Bekki. 2015. Higher-order logical inference with compositional semantics. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2055–2061, Lisbon, Portugal. Association for Computational Linguistics.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666, Portorož, Slovenia. European Language Resources Association (ELRA).

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo

Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. 2020. Universal Dependencies v2: An evergrowing multilingual treebank collection. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 4034–4043, Marseille, France. European Language Resources Association.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Siva Reddy, Oscar Täckström, Slav Petrov, Mark Steedman, and Mirella Lapata. 2017. Universal semantic parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 89–101, Copenhagen, Denmark. Association for Computational Linguistics.

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

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 2010)*, pages 1083–1091, Beijing, China. Coling 2010 Organizing Committee.

Sumire Uematsu, Takuya Matsuzaki, Hiroki Hanaoka, Yusuke Miyao, and Hideki Mima. 2013. Integrating multiple dependency corpora for inducing wide-coverage Japanese CCG resources. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1042–1051, Sofia, Bulgaria. Association for Computational Linguistics.

Masashi Yoshikawa, Hiroshi Noji, and Yuji Matsumoto. 2017. A* CCG parsing with a supertag and dependency factored model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 277–287, Vancouver, Canada. Association for Computational Linguistics.

Masashi Yoshikawa, Hiroshi Noji, Koji Mineshima, and Daisuke Bekki. 2019. Automatic generation of high quality CCGbanks for parser domain adaptation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 129–139, Florence, Italy. Association for Computational Linguistics.

A Appendix

Treebank	Conversion Rate		Statistics							Coverage (dev)			Coverage (test)		
	#Sent.	%Converted	%Cross.	#Tok.	#Cat.	#Cat.>1	#Cat./Tok.	#Rules	#U. Rules	%Lex.	%Sent.	%Rule	%Lex.	%Sent.	%Rule
Ancient_Greek-PROIEL	17080	55.56	37.52	16478	504	317	1.76	78222	795	91.06	68.16	99.40	90.79	69.43	99.62
Ancient_Greek-Perseus	13919	35.76	62.38	13655	442	278	1.54	43467	799	91.78	42.12	95.74	93.14	51.25	95.97
Armenian-ArmTDP	2502	82.37	7.15	11413	467	276	1.62	37669	967	90.74	51.23	98.13	90.32	44.21	98.61
Basque-BDT	8993	65.25	31.52	18228	433	263	1.80	64082	829	92.54	60.39	99.09	93.00	63.44	98.87
Belarusian-HSE	25231	83.10	5.08	46768	568	361	1.61	217110	1165	94.31	75.81	99.81	93.83	70.96	99.59
Bulgarian-BTB	11138	92.17	3.06	27506	358	244	1.62	130639	676	94.97	73.29	99.57	95.53	76.49	99.76
Buryat-BDT	927	89.54	8.20	3871	164	102	1.38	8279	340	_	_	_	73.47	28.01	64.94
Catalan-AnCora	16678	72.30	5.57	28626	764	462	2.14	340561	1374	91.24	57.38	99.83	90.97	60.48	99.74
Chinese-GSD	4997	76.99	2.24	16602	516	303	1.94	92432	937	90.07	42.82	99.54	89.83	45.14	99.66
Chinese-GSDSimp	4997	78.37	0.02	16841	519	311	1.94	94353	934	90.11	43.64	99.53	89.95	45.24	99.67
Classical_Chinese-Kyoto	55514	99.04	0.01	7920	327	229	4.22	221786	504	78.36	86.05	99.89	78.98	86.74	99.85
Coptic-Scriptorium	1873	72.98	13.24	2179	242	160	2.85	31056	357	85.59	51.70	99.01	85.04	52.08	99.43
Croatian-SET	9010	80.81	8.47	32938	628	389	1.74	147870	1177	93.41	55.02	99.56	93.17	57.55	99.54
Czech-CAC	24709	73.85	12.71	55370	1089	668	1.68	308130	2213	93.58	63.98	99.63	92.46	62.40	99.67
Czech-FicTree	12760	82.12	11.40	25135	676	393	1.76	110793	1293	91.65	67.99	99.37	91.14	67.24	99.12
Czech-PDT	87913	79.79	11.49	123512	1924	1146	2.18	1024644	4094	90.01	65.55	99.78	90.22	65.97	99.79
Danish-DDT	5512	68.03	21.35	13168	711	372	1.72	52217	1189	91.52	55.58	98.46	90.94	52.77	98.36
Dutch-Alpino	13603	80.69	14.33	24389	550	364	1.68	142284	878	91.40	62.70	99.72	90.33	59.48	99.67
Dutch-LassySmall	7341	90.98	6.05	14612	322	223	1.69	75899	550	91.41	66.84	99.45	92.45	75.40	99.62
English-EWT	16621	89.58	3.29	21211	504	319	2.15	197299	918	89.50	75.05	99.75	89.53	75.58	99.87
English-GUM	7402	86.81	4.80	14543	367	220	1.95	101034	702	91.07	66.05	99.68	91.88	68.74	99.76
English-LinES	5243	86.42	8.07	9633	405	253	2.21	72267	772	90.37	63.12	99.34	90.78	64.36	99.65
English-ParTUT	2090	88.52	1.82	6922	277	183	1.78	39614	450	92.34	52.78	99.46	91.54	54.17	99.63
Estonian-EDT	30972	91.13	3.22	80980	1154	723	1.86	365722	2428	89.97	55.16	99.60	89.45	53.22	99.67
Estonian-EWT	5536	88.31	4.28	15051	444	286	1.76	52672	844	90.59	59.31	99.27	90.32	54.00	98.69
Faroese-FarPaHC	1621	69.83	0.19	2852	393	218	2.56	22681	755	84.84	25.50	96.95	86.89	34.20	97.42
Finnish-FTB	18723	90.36	7.70	42166	440	306	1.56	122110	971	93.16	75.60	99.54	92.95	71.85	99.65
Finnish-TDT	15136	87.15	6.14	50632	759	465	1.56	156264	1562	92.81	62.61	99.52	92.27	61.34	99.56
French-GSD	16341	89.24	4.06	41377	485	325	1.62	330766	880	94.76	71.94	99.88	94.25	66.85	99.76
French-ParTUT	1020	81.67	5.10	3627	219	136	1.66	19660	337	91.74	58.59	98.54	95.11	64.65	98.76
French-Sequoia	3099	90.32	2.13	9063	290	195	1.87	56401	490	93.98	68.48	99.50	93.94	65.30	99.56
French-Spoken	2837	82.23	9.02	3732	367	210	2.26	22313	499	90.29	64.03	98.46	90.27	62.07	98.38
Galician-TreeGal	1000	71.60	11.20	4023	204	130	1.54	14340	366	_	_	_	95.44	58.30	98.53
German-GSD	15590	84.33	9.30	44634	615	380	1.54	219328	1123	90.62	60.03	99.53	90.28	59.60	98.98
German-HDT	189928	86.65	6.76	173381	1380	921	1.99	2590192	2202	92.41	82.46	99.96	92.58	82.59	99.96
Gothic-PROIEL	5401	72.38	17.57	6071	315	190	2.10	28428	489	88.88	63.44	98.60	91.30	70.68	99.21
Greek-GDT	2521	88.62	5.63	10927	325	216	1.71	51344	613	94.78	59.67	99.45	94.67	62.09	99.42
Hungarian-Szeged	1800	74.67	21.11	10290	289	182	1.46	28629	649	95.24	50.00	98.48	95.29	54.76	98.74
Icelandic-IcePaHC	44029	69.69	0.37	47189	1603	927	2.34	505869	3248	84.33	51.38	99.48	86.62	51.89	99.44
Icelandic-Modern	6928	56.78	0.38	6153	399	397	2.61	63630	746	93.65	84.45	99.50	92.93	78.51	99.48
Indonesian-CSUI	1030	88.54	1.84	4492	282	176	2.15	23730	437	_	_	_	87.89	32.83	98.90
Indonesian-GSD	5593	80.05	0.97	18011	437	251	1.82	82675	880	91.52	56.10	99.55	91.28	58.85	99.29
Irish-IDT	4910	45.21	15.05	8519	362	214	1.67	36505	628	90.93	54.71	99.05	90.29	60.59	97.92
Italian-ISDT	14167	92.07	1.36	27493	592	363	1.75	245285	1011	93.60	68.76	99.74	94.45	70.23	99.87
Italian-ParTUT	2090	89.47	2.01	8165	301	182	1.60	45495	478	93.16	58.74	97.56	95.36	66.91	99.02
Italian-TWITTIRO	1424	72.61	1.05	5394	230	148	1.59	21240	417	90.87	37.86	99.09	91.68	42.72	98.79
Italian-VIT	10087	85.37	3.48	22299	705	415	1.93	208443	1190	89.53	46.79	99.88	90.03	57.27	99.58
Japanese-GSD	8100	89.02	0.32	19689	227	155	1.84	163124	414	94.24	66.45	99.90	93.66	65.94	99.92
Kazakh-KTB	1078	88.96	7.33	4000	151	96	1.28	8283	327	_	_	_	89.93	61.65	84.52
Korean-Kaist	27363	73.61	21.70	73374	1187	687	1.61	241994	2498	91.76	59.43	99.48	91.29	53.06	99.41
Kurmanji-MG	754	72.02	8.49	2229	155	93	1.40	6442	256	_	_	_	88.77	25.38	80.44
Latin-ITTB	26977	58.25	36.26	13736	623	386	2.44	200633	1252	85.89	73.15	99.77	84.52	74.05	99.68
Latin-LLCT	9023	69.46	28.86	5601	397	284	2.07	102255	727	90.88	85.26	99.45	89.97	87.48	99.69
Latin-PROIEL	18411	60.63	28.39	16757	435	290	1.83	74966	763	90.30	70.30	99.39	91.10	74.59	99.44
Latin-Perseus	2273	49.05	48.13	4597	217	131	1.36	9597	397	_	_	_	95.48	67.36	97.01
Latin-UDante	1721	38.76	48.17	5045	302	185	1.67	15982	629	92.92	32.70	97.22	90.31	26.52	97.06
Latvian-LVTB	15351	83.71	6.65	43753	672	419	1.77	185104	1395	91.12	58.56	99.63	91.04	57.83	99.61
Lithuanian-HSE	263	76.81	14.07	1815	159	92	1.39	3753	338	90.81	42.50	90.51	92.29	37.78	92.84
Livvi-KKPP	125	84.00	12.80	652	73	51	1.19	1101	140	_	_	_	80.84	25.00	67.61
Maltese-MUDT	2074	84.62	3.86	7385	233	153	1.83	33081	448	92.30	50.72	99.47	92.35	49.17	99.16
Marathi-UFAL	466	93.13	4.08	897	84	49	1.55	3176	166	93.92	70.45	98.94	89.47	65.91	95.78

Treebank	Conversion Rate		Statistics							Coverage (dev)			Coverage (test)		
	#Sent.	%Converted	%Cross.	#Tok.	#Cat.	#Cat.>1	#Cat./Tok.	#Rules	#U. Rules	%Lex.	%Sent.	%Rule	%Lex.	%Sent.	%Rule
North_Sami-Giella	3122	92.60	4.39	7398	212	141	1.52	21054	414	_	_	_	93.60	61.35	97.60
Norwegian-Bokmaal	20044	82.65	7.39	30539	693	408	1.81	218696	1204	92.18	73.58	99.68	91.73	72.23	99.78
Norwegian-Nynorsk	17575	80.29	7.71	28516	720	423	1.81	205739	1209	91.76	70.75	99.66	92.05	72.13	99.73
Old_Church_Slavonic-PROIEL	6338	74.77	16.31	7138	299	198	2.01	31777	471	91.24	73.16	99.03	91.06	72.23	99.14
Old_East_Slavic-RNC	957	63.22	30.72	4360	215	136	1.38	12165	441	_	_	_	94.38	43.39	94.45
Old_East_Slavic-TOROT	16944	77.82	15.20	23770	467	297	1.63	87124	765	92.70	78.53	99.57	93.35	74.72	99.58
Old_French-SRCMF	17678	82.29	15.06	15417	535	321	2.20	110552	787	87.24	72.76	99.77	88.34	74.35	99.70
Persian-PerDT	29107	80.58	14.22	28743	629	400	2.35	345299	1171	92.53	74.05	99.87	92.04	73.36	99.85
Persian-Seraji	5997	76.84	5.45	12507	601	345	2.51	99863	1022	88.65	48.18	99.58	86.87	40.69	99.67
Polish-LFG	17246	98.24	0.64	32501	369	262	1.51	112232	571	95.22	83.60	99.78	93.99	76.98	99.38
Polish-PDB	22152	85.50	6.25	56756	790	483	1.66	264893	1607	94.28	70.60	99.68	94.14	70.75	99.67
Portuguese-Bosque	9364	72.86	18.25	21552	436	281	1.60	132130	791	94.38	73.01	99.75	94.61	70.39	99.62
Portuguese-GSD	12078	83.73	5.20	29470	624	374	1.88	242474	1127	93.41	64.24	99.87	93.85	65.53	99.81
Romanian-Nonstandard	26225	84.10	5.43	31821	1104	676	2.20	439787	2035	89.39	67.48	99.87	88.38	59.31	99.59
Romanian-RRT	9524	82.38	8.82	30510	641	416	1.73	169375	1153	93.29	55.23	99.73	94.42	63.71	99.76
Romanian-SiMoNERo	4681	77.57	14.61	15734	370	249	1.86	101366	688	91.96	52.87	99.73	93.12	56.86	99.78
Russian-GSD	5030	90.14	6.12	27675	384	252	1.44	80866	755	95.62	66.41	99.39	96.00	67.53	99.60
Russian-SynTagRus	61889	88.55	7.05	115101	1181	767	1.95	897395	2440	91.83	68.13	99.84	92.01	67.61	99.84
Russian-Taiga	17870	90.41	6.12	36352	541	332	1.51	150354	1069	93.60	70.05	99.61	93.65	72.76	99.72
Sanskrit-Vedic	3997	75.06	23.42	5386	239	151	1.63	15093	424	_	_	_	92.69	71.92	97.82
Scottish_Gaelic-ARCOSG	3798	55.32	7.11	3722	286	172	2.14	21447	432	89.48	64.12	98.69	86.06	58.51	98.17
Serbian-SET	4384	87.11	3.19	17933	403	258	1.70	79033	753	94.34	58.12	99.51	94.26	59.52	99.52
Slovak-SNK	10604	91.94	3.27	26470	490	321	1.49	85921	903	96.00	74.36	99.43	95.53	72.86	99.08
Slovenian-SSJ	8000	83.06	12.00	29524	415	280	1.54	105267	678	94.41	62.77	99.66	94.24	64.06	99.65
Slovenian-SST	3188	88.55	4.49	4955	288	180	1.93	20231	423	_	_	_	89.53	66.50	98.46
Spanish-AnCora	17680	79.00	5.57	36064	874	566	2.11	392705	1584	91.31	54.32	99.81	91.09	55.87	99.76
Spanish-GSD	16013	82.56	5.85	42462	750	439	1.68	318965	1320	93.41	63.55	99.78	93.37	58.60	99.86
Swedish-LinES	5243	88.08	5.61	13258	524	318	1.93	70357	921	90.89	55.39	98.99	91.61	60.69	99.43
Swedish-Talbanken	6026	91.74	2.99	15156	478	299	1.83	79649	770	88.39	46.28	99.35	90.67	57.19	99.22
Tamil-TTB	600	97.83	1.67	3515	244	144	1.67	9079	435	89.71	45.57	95.19	89.46	31.90	95.70
Telugu-MTG	1328	99.77	0.15	2046	92	63	1.43	5410	163	96.10	91.60	98.02	96.61	91.03	99.00
Turkish-BOUN	9761	88.85	3.34	33475	628	358	1.61	98665	1523	91.38	61.14	99.11	91.58	59.91	99.27
Turkish-FrameNet	2698	96.85	0.26	8155	154	101	1.36	17020	276	95.20	81.31	99.44	94.74	77.11	99.10
Turkish-IMST	5635	88.61	6.35	16247	520	301	1.73	43696	1148	92.09	60.00	98.47	91.73	64.34	98.34
Turkish-Kenet	18687	93.40	2.23	46523	586	343	1.71	157843	1371	90.54	55.80	99.53	91.07	57.55	99.62
Turkish-Penn	9557	95.47	1.31	21467	422	256	1.68	76148	844	88.68	55.23	99.57	90.46	65.65	99.68
Turkish-Tourism	19749	98.72	0.51	4898	202	140	2.56	74151	394	89.61	91.71	93.63	87.52	88.92	99.87
Turkish_German-SAGT	2184	77.20	13.42	5684	339	208	2.15	24823	557	91.49	36.28	97.29	91.53	35.98	97.81
Ukrainian-IU	7060	87.31	7.69	28985	493	299	1.52	94522	1038	94.86	62.67	99.49	94.63	64.60	99.26
Upper_Sorbian-UFAL	646	82.82	11.30	3746	149	99	1.26	8155	299	_	_	_	93.01	43.60	90.23
Uyghur-UDT	3456	91.38	4.98	11007	288	174	1.71	34969	708	93.41	59.98	98.37	93.17	58.55	97.66
Welsh-CCG	1833	49.37	1.91	3688	153	97	1.81	13904	289	95.96	58.29	98.23	95.76	57.28	98.15
Western_Armenian-ArmTDP	1780	81.85	9.72	8383	353	205	1.61	25019	750	95.01	54.91	97.31	94.33	47.49	97.64
Wolof-WTB	2107	83.91	2.99	5227	361	214	2.23	33500	594	87.90	39.33	98.83	87.72	41.86	98.61

Table 4: Conversion results on 105 treebanks of 65 languages in UD v2.8. Column names from left to right: (1) Treebank, (2) Number of sentences, (3) Conversion rate, (4) Percentage of sentences with cross-serial dependencies, (5) Number of distinct tokens, (6) Number of distinct categories, (7) Number of distinct categories that appear more than once, (8) Average number of categories per token, (9) Number of CCG rule instantiations, (10) Number of unique CCG rules, (11) Lexical coverage on `dev`, (12) Sentential coverage on `dev`, (13) Syntactic rule coverage on `dev`, (14) Lexical coverage on `test`, (15) Sentential coverage on `test`, (16) Syntactic rule coverage on `test`.

Treebank	#Train samples	#Test samples	PARSEVAL Unlabelled			PARSEVAL Labelled			Supertagging accuracy
			%Precision	%Recall	%F1	%Precision	%Recall	%F1	
Belarusian-HSE	18878	947	89.21	73.27	80.46	69.76	57.30	62.92	82.19
Catalan-AnCora	9511	1341	92.64	76.84	84.00	80.21	66.52	72.73	88.23
Cls_Chinese-Kyoto	45315	4412	94.72	94.20	94.46	72.62	72.21	72.41	79.59
Czech-PDT	54698	8090	92.90	86.79	89.74	77.08	72.01	74.46	86.69
English-EWT	11116	1929	92.74	88.63	90.64	79.25	75.74	77.45	86.76
Estonian-EDT	22467	2920	89.64	75.04	81.69	65.31	54.67	59.52	75.41
Finnish-FTB	13538	1705	88.73	79.04	83.6	62.07	55.29	58.49	69.33
French-GSD	12890	359	93.16	81.00	86.65	82.22	71.49	76.48	90.13
German-HDT	132361	16104	96.25	94.15	95.19	88.38	86.45	87.40	94.74
Icelandic-IcePaHC	24363	3386	92.28	67.49	77.96	70.85	51.81	59.85	76.79
Italian-ISDT	12094	440	91.73	85.74	88.63	79.48	74.29	76.80	88.48
Korean-Kaist	16839	1764	91.59	72.91	81.19	61.63	49.06	54.63	78.72
Latin-ITTB	13114	1318	93.38	80.02	86.18	73.08	62.62	67.45	83.44
Norwegian-Bokmaal	12957	1595	93.37	86.92	90.03	78.75	73.31	75.93	86.39
Old_French-SRCMF	11428	1622	92.47	80.66	86.16	68.73	59.94	64.04	76.12
Old_East_Slavic-TOROT	10400	1425	88.82	73.51	80.44	54.89	45.44	49.72	72.28
Persian-PerDT	21109	1186	94.69	90.54	92.57	81.91	78.32	80.08	89.55
Polish-PDB	15144	1904	91.90	79.79	85.42	72.06	62.57	66.98	82.12
Romanian-Nonstandard	20183	924	90.58	55.37	68.72	67.25	41.11	51.02	83.01
Russian-SynTagRus	43271	8898	92.82	77.30	84.35	76.14	63.41	69.19	88.98
Spanish-AnCora	11283	1389	92.15	72.62	81.23	78.57	61.92	69.26	87.33
Turkish-Tourism	15173	2147	98.05	97.71	97.88	74.23	73.97	74.10	81.25

Table 5: CCG parsing performance measured on the converted test sets of 22 treebanks of 22 languages that have more than 10,000 sentences in the training sets.