# Converting a Dependency Treebank to a Categorial Grammar Treebank for Italian

Johan Bos
Bologna, Italy
bos@meaningfactory.com

Cristina Bosco
Università Torino
bosco@di.unito.it

Alessandro Mazzei
Università Torino
mazzei@di.unito.it

**Abstract**

The Turin University Treebank (TUT) is a treebank with dependency-based annotations of 2,400 Italian sentences. By converting TUT to binary constituency trees, it is possible to produce a treebank of derivations of Combinatory Categorial Grammar (CCG), with an algorithm that traverses a tree in a top-down manner, employing a stack to record argument structure, using Part of Speech tags to determine the lexical categories. This method reaches a coverage of 77%, resulting in a CCGbank for Italian comprising 1,837 sentences, with an average length of 22,9 tokens. The CCGbank for English has proven to be a useful tool for developing efficient wide-coverage parsers for semantic interpretation, and the Italian CCGbank is expected to be an equally useful linguistic resource for training statistical parsers.

## 1 Introduction

Treebanks have played an important role in the development of robust parsers exploring statistical methods to achieve wide coverage. The key example in this tradition is the Penn Treebank [10], a large collection of English sentences taken from the Wall Street Journal annotated with syntactic structures. The aim of this article is to present the first version of an Italian treebank based on categorial grammar, translated from an existing manually crafted dependency-based treebank [4]. We focus on the various translation steps required to achieve a high-quality treebank.

Our treebank is based on CCG, combinatory categorial grammar [12], a lexicalised grammar formalism encoding all non-local dependencies in its lexical categories. A further motivation for using CCG is its transparency between syntactic categories and semantic types, providing an ideal platform for automatically building formal semantic representations [2]. Other treebanks based on categorial grammar have been developed in the past, of which the English CCGbank [7] derived from the Penn Treebank [10] is the prime example. The English CCGbank has proven to be a useful resource for training robust parsers [6], and we follow its design as closely as possible.

For automatically deriving Italian categorial grammar, previous work on has been carried out by [1], who provide a method for translating TUT dependency structure to derivations of type-logical grammar, but only for a relatively small set of examples with low structural complexity, producing a lexicon of 1,909 words based on 400 derivations with an average of two categories per words. Even though our work is similar in spirit, we also deal with more complex cases extending the coverage considerably.

# 2 Background

## 2.1 The Turin University Treebank (TUT)

The starting point of our conversion in CCG is TUT, the Turin University Treebank[1]. This Italian treebank currently includes 2,400 sentences corresponding to around 72,150 tokens. This annotated corpus consists of three parts: 1,100 sentences from newspaper texts (mainly from *La Stampa* and *La Repubblica*), 1,100 sentences from the Italian Civil Law Code, and 200 sentences from the Italian section of the JRC-Acquis corpus.[2]

The development of TUT has its roots in a dependency-based annotation following the major principles of Word Grammar [8]. Central to this is a notion of argument structure described by a rich set of grammatical relations that can include three components: morpho-syntactic, functional-syntactic, and semantic information [3]. The TUT annotation process includes automatic POS tagging and parsing [9], completed with a set of automatic and manual correctness and consistency checks.

To promote applications of TUT, several efforts have been directed to automatically converting the treebank into other formats. Among these are bracketed labelling known from the Penn Treebank [10], as well as the Xbar-like format called Constituency TUT (henceforth ConsTUT), which forms an important step in converting TUT to CCG. These mappings to other formats have not only increased the comparability with other existing linguistic resources, but also improved the quality of the annotated material. Even though the size of TUT is relatively small compared to the Penn Treebank, the EVALITA 2009 shared task on parsing showed that dependency parsers trained on TUT are close to the state-of-the-art [5].

## 2.2 Combinatory Categorial Grammar (CCG)

CCG is a lexicalised theory of grammar in which all syntactic dependencies are encoded in the lexical categories [12]. The version of CCG that we adopt is based on the English CCGbank [7], comprising a set of CCG derivations derived from the Wall Street Journal texts from the Penn Treebank [10].

[1] See `http://www.di.unito.it/~tutreeb/`, [4].

[2] See `http://langtech.jrc.it/JRC-Acquis.html`.

The basic categories are `s` (sentence), `n` (noun), `pp` (prepositional phrase), `np` (noun phrase) and `t` (text). Functor categories are composed out of the basic categories with the help of slashes indicating order and position of arguments: a functor category $\alpha\backslash\beta$ yields a category $\alpha$ when it finds an argument of category $\beta$ on its left, and a functor category $\alpha/\beta$ yields a category $\alpha$ when it finds an argument of category $\beta$ on its right. We follow the convention introduced in CCGbank and associate the category `s` with a feature indicating sentence mood or aspect of verb phrases (Fig. 1).

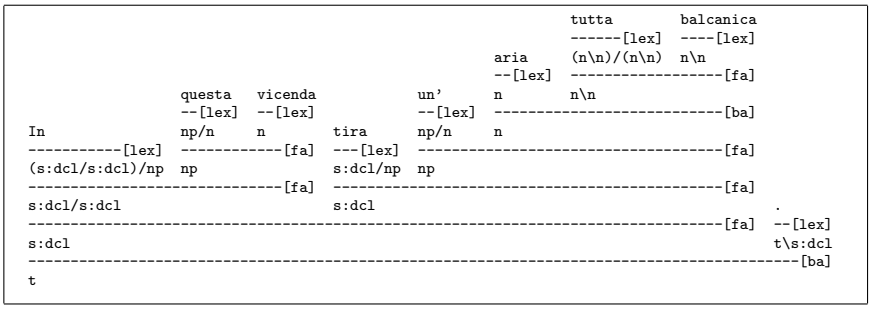


Figure 1: CCG derivation for *In questa vicenda tira un' aria tutta balcanica.*

To combine categories deriving new categories, CCG is equipped with a small set of combinatory rules and a couple of non-combinatory rules. The combinatory rules combine two categories and produce a new one. They comprise forward application (>), backward application (<), forward and backward composition (>B and <B), forward and backward substitution (>S and <S), their crossing variants, and generalised versions of the composition rules. All of these rules have a direct semantic interpretation, and give expressive power that go beyond context free grammars [12].

The non-combinatory rules consist of the type-raising and type-changing rules. They are unary rules, mapping a single category into a new one. The type-raising rules (>T and <T) change an argument in a functor in a systematic way. The type-changing rules are used for covering elliptical expressions, such as pro-drop, adjective-like participles, and determinerless noun phrases.

# 3 Method

We take as input a set of sentences in the TUT dependency format. These are first mapped onto constituency trees, then transformed into binary trees, and finally converted into CCG derivations.

### 3.1 From Dependency Structures to Constituency Trees

ConsTUT is a constituency-based annotation with constituents decorated by TUT relations. In ConsTUT trees each terminal category X corresponds to a node (i.e. token) of a TUT tree, and projects into non-terminal nodes which represent intermediate (Xbar) and maximal (XP) projections of X, following Xbar theory [11]. Each node is classified as either head (h), argument (a), or modifier (m). To illustrate this projection of categories, consider for instance the adverb *tutta* in Fig. 2. This adverb projects on ADVbar and then on ADVP, as the ConsTUT tree in Fig. 3 shows.

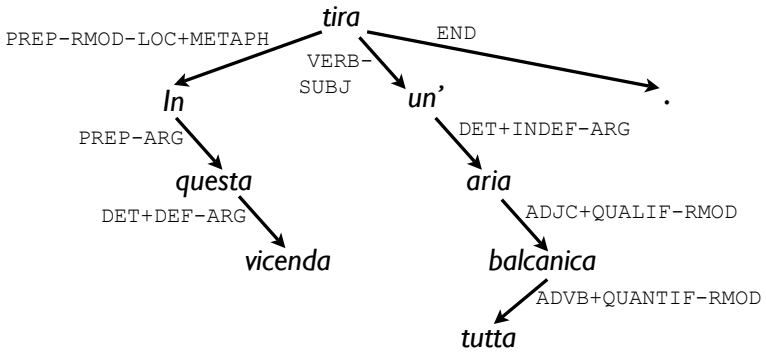


Figure 2: TUT A-6, *In questa vicenda tira un' aria tutta balcanica.*

Further following Xbar theory, the distinction between arguments and modifiers is structurally marked. Arguments, usually closer to their head, are daughters of intermediate projections and sisters of terminal categories. Modifiers, on the other hand, are both sisters and daughters of intermediate projections. Schematically, this can be viewed as: [XP (Xbar (Xbar (X)(ARG)) (MOD))].

The conversion algorithm from TUT to ConsTUT is adapted from [13], who employed it for the conversion of dependency structures in Penn Treebank style phrase structures, i.e., constituents featuring a minimal projection strategy. The input of our algorithm are ordered dependency trees, that is projective structures where dependents of the same head are ordered according to their positions in the sentence. Explicit marking of null elements can occur to resolve cases of non-projectivity too. The output of the algorithm are constituency trees applying a maximal projection strategy.

The main information that is present in a constituency tree, but not in a dependency tree, is the type of non-terminal nodes (e.g. NP, VP and S). Therefore, the major goal of the algorithm is to recover the types of phrases that each node of the dependency tree projects. In other words, the task here is to find the expansions in constituency terms of the grammatical category of terminal nodes, which have to be annotated as non-terminal nodes in the

```
h(S
    a(DP-SUBJ*[533])
    h(Sbar
         h(VP
             h(Vbar
                m(PP-RMOD-LOC+METAPH
                    h(PREP 1/In)
                    a(DP-ARG
                       h(Dbar
                          h(ADJ~DE 2/questa)
                          a(NP-ARG
                              h(Nbar
                                  h(NOU~CS 3/vicenda))))))
                h(Vbar
                    h(VMA~RE 4/tira)
                    a(DP-EXTRAPOSEDSUBJ*533
                        h(Dbar
                            h(ART~IN 5/un')
                            a(NP-ARG
                                h(Nbar
                                    h(Nbar h(NOU~CS 6/aria))
                                    m(ADJP-RMOD
                                        m(ADVP-RMOD
                                            h(ADVbar h(ADVB 7/tutta)))
                                        h(ADJbar
                                            h(ADJ~QU 8/balcanica)))))))))
    m(PUNCT-END 9/.))
```

Figure 3: ConsTUT tree for *In questa vicenda tira un' aria tutta balcanica.*

constituency trees. The trees also contain Part of Speech (POS) tags on terminal nodes, using a simplified tagset of that used in TUT [3].

Grammatical categories can interact in dependency trees. To build a corresponding constituency tree, one needs to know how one can represent this interaction in constituency terms, that is, how grammatical categories and their projections combine in constituency structure. This is governed by language specific constraints, depending on the types of modifiers and arguments that a head can take and their positions related to the head itself. All this information is encoded in a look-up table that the converter exploits.

### 3.2 From Constituency Trees to Binary Trees

The lexical categorisation algorithm, which is last in the pipeline, requires binary trees (trees with at most two branches) as input. The syntactic analyses of ConsTUT are represented by $n$-ary trees, but not necessarily binary trees (see Fig. 3). Given the distinction between head, modifier and argument, there are four possible kinds of trees (Fig. 4).

To map a tree with $n$ ($n > 2$) branches into a tree with $n - 1$ branches, we select a HEAD-X or X-HEAD sequence from the ordered list of branches (where X is ARG or MOD), and replace it by a new branch marked by HEAD forming a binary tree of the two selected branches. This process is iterated

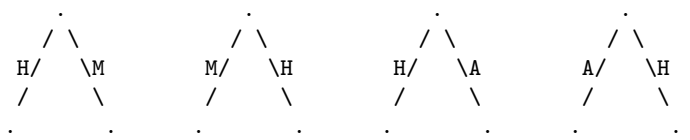

Figure 4: Possible binary sub-trees.

for each node until it is completely binary. The procedure is illustrated for a ternary tree mapped to a binary tree in Figure 5.

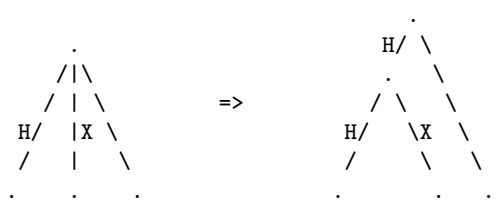

Figure 5: Producing binary trees.

This is a general rule — there are specific rules that cover awkward cases such as punctuation. At this stage we also deal with Italian "definite prepositions" such as *sulla*, which are contractions of a preposition and a definite article. In ConsTUT these appear as two different nodes (PREP and ART-DE) in the tree, related to each other by co-indexing. These two nodes are mapped onto a newly marked node (DEFPREP) in order to distinguish it from ordinary prepositions in the categorisation process.

### 3.3 From Binary Trees to Categorial Grammar Derivations

The categorisation algorithm takes a binary constituency tree as input and produces a CCG derivation. The core of the algorithm is based on "pure" categorial grammar, using only forward application (>) and backward application (<) of the set of combinatory rules. In a nutshell, the algorithm proceeds as follows: it traverses the binary input tree in a top-down fashion, starting with the root node. The final values for the categories for the nodes, however, are generated via a bottom-up strategy, determined by the POS tags. The algorithm makes use of a stack on which it pushes the categories of arguments encountered in the binary tree. The elements of the stack determine the lexical categories of heads.

There are four general rules that deal with the standard cases in Fig. 4, which can be overridden by more specific rules to cover special linguistic constructions, such as punctuation or coordination. First consider the two modifier cases, where the algorithm produces a CCG derivation for the head first. Suppose this yields category X, then we make the category for the modifier X\X in the HEAD-MOD case, introducing the backward application rule (<). The MOD-HEAD case forms a mirror image of this mapping, yielding

the category X/X for the modifier by virtue of the forward application rule (>). Note that, in both cases, nothing is altered to the value of the stack (S); it is just passed on when translating HEAD (Fig. 6).

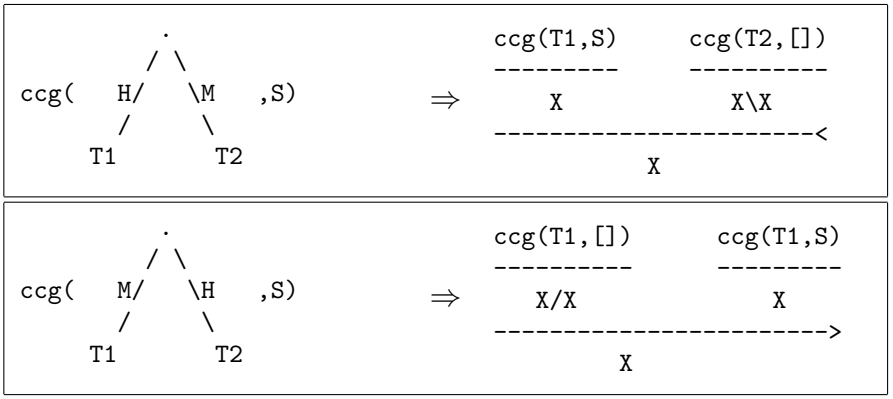


Figure 6: The modifier case for translating binary trees to CCG.

In the HEAD-ARG and ARG-HEAD cases, the argument is analysed first. The resulting category is pushed on the stack (plus its direction: \ or /). Suppose that the translation of ARG yields the category Y, then the HEAD-ARG case maps to a forward application rule with X/Y being the category for the head constituent. The ARG-HEAD case is a mirror image of the HEAD-ARG translation rule. Here we introduce the backward application rule with X\Y being the category for HEAD, and the category produced for ARG is pushed onto the stack. This is illustrated by Fig. 7.

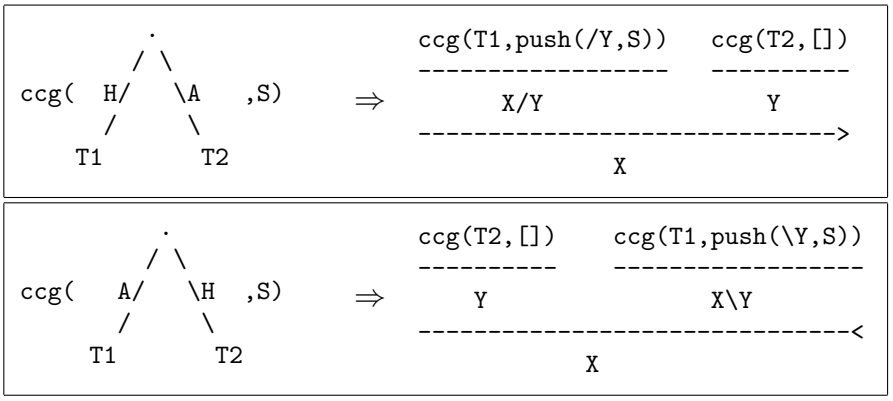


Figure 7: The argument case for translating binary trees to CCG.

These four general rules are used in traversing the binary tree in a top-down fashion. Once a leaf node is reached, the lexical categories are determined, which in turn provide information to determine the values of the non-lexical categories encountered earlier. The mapping from a leaf node to a lexical CCG category depends on whether it is playing the role of head,

argument, or modifier. A leaf node of type MOD is either of the form X/X or X\X, where X is already determined by the head using either the HEAD-MOD or MOD-HEAD rule. In the case of ARG, the lexical category is determined by the assigned Part of Speech in TUT (see Table 2). For HEAD we follow the same strategy as for ARG, but we resort to the stack to determine the number and direction of arguments. The features introduced on the category `s` are shown in Table 1.

Passive sentences are marked in ConsTUT by empty argument nodes marked as logical subject of a verb phrase. These are pushed onto the stack as were they ordinary arguments. However, when generating a lexical category for a verb phrase, the feature chosen for `s` will be `pss` when a logical subject is a member of the stack. Coordination is dealt with by giving conjunctors the category (X\X)/X, where X can be any category.

Finally, a post-processing procedure deals with empty nodes, clitics, normalisation of accents, and certain kinds of punctuation. Empty nodes, such as Italian pro-drop, introduce type-changing rules in the CCG derivation. Verbal clitics are explicitly marked as arguments combined with an application rule. Opening and closing parentheses and quotes are given categories (X/p)/X and X\(X/p), respectively, where X can be any category. We also normalize the accents post-processing step, since TUT allows for distinct ways of encoding Italian accents.

## 4 Results

Recall that the conversion process from TUT dependencies to CCG derivations forms a pipeline of three components. These conversion steps are all performed automatically, and hence prone to errors. Reasons for failure in the conversion are complex cases of coordination, clitics, verbal structures, or elliptical phrases. In part these can and will be dealt with in future, re-

Table 1: Features on category `s` (verb phrases and sentences).

| Part of Speech | Lexical Category | Description |
|---|---|---|
| VMA-GE | `s:ger` | gerund |
| VMA-PA | `s:pap` | past participle |
| VMA-PE | `s:prp` | present participle |
| VMA-RE | `s:dcl` | present |
| VMA-IN | `s:inf` | infinitive |
| VMA-IP | `s:imp` | imperative |
| | `s:adj` | adjective phrase |
| | `s:pss` | passive |
| | `s:ynq` | yes/no-question |
| | `s:whq` | wh-question |

Table 2: Mapping from POS tags to lexical categories.

| **Part of Speech** | **Lexical Category** | **Description** |
|---|---|---|
| VMA | `s` | main verb |
| VMO | `s` | modal verb |
| VAU | `s` | auxiliary verb |
| ART | `np` | article |
| PRDT | `np` | pre-determiner |
| PRO | `np` | pronoun |
| PRO-PO | `np`, `n` | possessive pronoun |
| NOU | `n`, `np` | common noun |
| ADJ | `s:adj\np`, `n/n`, `n\n` | adjective |
| PREP | `pp`, X/X, X\X | preposition |
| NUMR | `n`, `np` | numeral |
| PUNCT-END | `t` | punctuation |
| CONJ | X | conjunction |

vised versions of the treebank, by adding more specific transformation rules to the algorithm.

In Table 3 we report coverage of the conversion process by the number of well-formed trees that we obtain as output after each processing step. The civil law corpus yields the highest coverage, probably because it is a part of TUT containing many relatively short sentences, with less complex, legal language. This claim is confirmed by looking at the length of the sentences in the three corpora: the newspaper corpus has an average length of 24.04 (19,355 tokens on 807 sentences), the civil law corpus 20.97 (18,766 tokens on 896 sentences), and the JRC-Acquis corpus 29.08 (3,839 tokens on 132 sentences). Hence, it is likely that the JRC-Acquis corpus produces a high number of mistakes because the sentences are significantly longer. Moreover, this corpus is only recently been added to TUT, and as a consequence still contains a number of syntactic constructions that are not yet covered by the dependency to constituency conversion step.

Table 3: Number of trees in the three corpora after each processing step.

| **Corpus** | **TUT** | → | **ConsTUT** | → | **Bin** | → | **CCG** | |
|---|---|---|---|---|---|---|---|---|
| Newspaper | 1,100 | → | 890 | → | 827 | → | 807 | (73%) |
| Civil Law | 1,100 | → | 993 | → | 909 | → | 898 | (82%) |
| JRC-Acquis | 200 | → | 147 | → | 133 | → | 132 | (66%) |

The total number of different lexical categories generated for all three corpora is 1,152, of which 627 occur more than once. For comparison, the English CCGbank hosts 1,286 different lexical categories, of which 847 occur at least twice [7]. The ten most frequent categories are shown in Table 4,

and the average number of categories per token is 1.66.

Since one of the aims of the Italian CCGbank is to produce statistical language models, an important question to ask is whether the size of the treebank is large enough for this purpose. The size of TUT is relatively small compared to the size of the Penn Treebank, but it would be helpful to estimate the required size of a CCGbank as developing annotated tree structures is a costly business. We can't completely answer this question, but what we can do, is look at the number of different categories produced for each corpus to get a rough idea.

Table 4: Frequency of the ten most common categories.

| **Category** | **Newspaper** | **Civil Law** | **JRC-Acquis** | **Total** |
|---|---|---|---|---|
| n | 4,477 | 4,572 | 904 | 9,953 |
| np/n | 1,807 | 1,658 | 345 | 3,810 |
| (n\n)/n | 1,228 | 1,050 | 288 | 2,566 |
| n\n | 1,254 | 932 | 365 | 2,551 |
| pp/n | 707 | 945 | 255 | 1,907 |
| t\s:dcl | 720 | 593 | 122 | 1,435 |
| np | 604 | 645 | 50 | 1,299 |
| n/n | 706 | 377 | 128 | 1,211 |
| n/pp | 279 | 359 | 142 | 780 |
| s:dcl/s:dcl | 297 | 248 | 52 | 597 |

We do this in Table 5, where we show the number of different categories, as well as the number of different POS-category pairs. We also computed these values with (+) and without (−) features on these categories. As the table shows, the numbers differ substantially, clearly caused by the size of the respective corpora. Next, we investigated the relation between the number of lexical categories and the size of the treebank. A treebank with a good coverage would show an emerging plateau in the number of categories by an increase of the number of sentences considered.

Table 5: Number of POS and categories of the three corpora.

| | **Newspaper** | **Civil Law** | **JRC-Acquis** | **Total** |
|---|---|---|---|---|
| CAT (−) | 541 | 411 | 188 | 714 |
| CAT (+) | 841 | 657 | 291 | 1,154 |
| POS+CAT (−) | 826 | 640 | 277 | 1,121 |
| POS+CAT (+) | 1,121 | 876 | 362 | 1,576 |

As Fig. 8 shows, the growth of categories is clearly decreasing, yet rising. This is caused in part by the small size of the treebank, but we think it is also due to the absence of forward and backward (crossed) composition rules in the categorisation algorithm implemented so far. Adding such combinatory

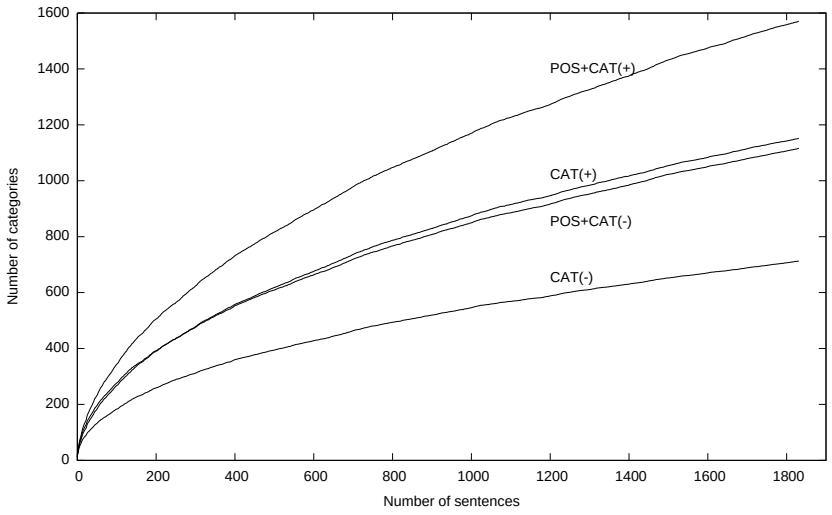


Figure 8: Growth of lexical categories with respect to number of sentences.

rules for specific but common linguistic structures is part of future work, as is abstracting over modifier and conjunction categories, to gain a further reduction of distinct lexical categories [7]. The current version of the Italian CCGbank is released under a creative commons license via `http://www.di.unito.it/~tutreeb/CCG-TUT/` in various formats: the derivations printed in human-readable format (as in Fig. 1), derivations printed as Prolog terms, or as token-POS-category tuples.

**Acknowledgements** We thank Jason Baldridge, James Curran, Leonardo Lesmo, Malvina Nissim, and Mark Steedman for their comments on earlier versions of this paper and their feedback on the Italian CCGbank.

# References

[1] R. Bernardi, A. Bolognesi, C. Seidenari, and F. Tamburini. Learning an italian categorial grammar. In R. Rossini Favretti, editor, *Frames, Corpora, and Knowledge Representation*, pages 185–200. 2008.

[2] J. Bos. Wide-Coverage Semantic Analysis with Boxer. In J. Bos and R. Delmonte, editors, *Semantics in Text Processing. STEP 2008 Conference Proceedings*, volume 1 of *Research in Computational Semantics*, pages 277–286. College Publications, 2008.

[3] C. Bosco. *A grammatical relation system for treebank annotation.* PhD thesis, University of Torino, 2004.

[4] C. Bosco, V. Lombardo, D. Vassallo, and L. Lesmo. Building a Treebank for Italian: a Data-driven Annotation Schema. In *Proc. 2nd Int. Conf. on Language Resources and Evaluation*, pages 99–105, Athens, 2000.

[5] C. Bosco, S. Montemagni, A. Mazzei, V. Lombardo, F. Dell'Orletta, and A. Lenci. Evalita'09 parsing task: comparing dependency parsers and treebanks. In *Proceedings of EVALITA 2009*, Reggio Emilia, 2009.

[6] S. Clark and J.R. Curran. Parsing the WSJ using CCG and Log-Linear Models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL '04)*, Barcelona, Spain, 2004.

[7] J. Hockenmaier. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. PhD thesis, Univ. of Edinburgh, 2003.

[8] R. Hudson. *Word Grammar*. Blackwell, Oxford and New York, 1984.

[9] L. Lesmo. The rule-based parser of the NLP group of the university of Torino. *Intelligenza Artificiale*, IV(2):46–47, 2007.

[10] M.P. Marcus, B. Santorini, and M.A. Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[11] A. Radford. *Syntactic theory and the structure of English. A minimalist approach*. Cambridge University Press, 1997.

[12] M. Steedman. *The Syntactic Process*. The MIT Press, 2001.

[13] F. Xia. *Automatic grammar generation from two different perspectives*. PhD thesis, University of Pennsylvania, 2001.