

Quality control of treebanks: documenting, converting, patching

Sabine Buchholz, Darren Green

Speech Technology Group, Cambridge Research Lab, Toshiba Research Europe Ltd
St George House, 1 Guildhall St, Cambridge CB2 3NH, United Kingdom
{sabine.buchholz, darren.green}@crl.toshiba.co.uk

Abstract

We report about our experiences with using many different syntactically annotated corpora (treebanks). We list various types of format and annotation errors we have noticed and propose common sense as well as novel ways to prevent, detect and handle these. We show how the quality of a treebank's annotation and its documentation are related and how the concepts of patching and versioning that come from the software community can be applied to treebanks in order to improve quality.

1. Introduction

The first author is one of the organizers¹ of the shared task on multilingual dependency parsing for the 2006 Conference on Computational Natural Language Learning, CoNLL-X.² For that task, treebanks for 13 different languages were converted to a common format (Buchholz et al., 2006). A further three (Sampson, 1995; Sima'an et al., 2001; Aduriz et al., 2003) were studied or partially converted but in the end not included in the task. So the shared task is easily the biggest "user" of treebanks to date. The converted treebanks have been made available to 28 interested groups and some bugs have already been reported. Through our work at Toshiba Research Europe we have experience with a further four treebanks that offer commercial licenses.

While the majority of treebanks are of high quality, we have noticed problems with some. As many of the treebanks are for languages we do not speak, we tend to notice issues with the form rather than with the content of the annotation. Such problems can often be detected by relatively simple means and are therefore **easily avoidable**. In this paper, we discuss the issues we encountered and suggest methods to detect them. In addition to quality of annotation, we want to draw attention to two other aspects of quality control. One is the quality of documentation of resources and the other is the handling of defects once they are noticed. We will show that these three aspects are related and how improvements in one can benefit the others.

2. Issues with treebanks

The issues we noticed range from pure format to more linguistic problems. While linguists using a treebank are normally only bothered by the latter, computational linguists should be concerned about the former as well, as they can directly affect the outcome of any experiment on the data.

2.1. End-of-line convention

Different operating systems use different control characters to terminate a line. We have encountered one treebank in which some files followed the Macintosh convention of carriage returns and others the UNIX convention of line feeds.

2.2. Whitespace

People who look at treebank annotations, including annotators, normally do not care much about whitespace. However, when faced with the task of converting a treebank from one format to another, whitespace becomes relevant. We cannot recall any treebank documentation that defined explicitly how whitespace is used. However, when looking at treebank files, it often becomes apparent that whitespace is used in a certain way. Our first version of a script for converting one of the treebanks to the shared task format made many assumptions about where spaces, tabs or blank lines were required or forbidden, based on our study of several treebank sentences. However, every single one of these assumptions was violated by at least one other sentence. From this we draw two conclusions: First, scripts taking a treebank as input should never make such assumptions, as they would introduce errors in the output. Second, treebank providers should explicitly mention any rules about whitespace that should hold, and then ensure that they do. We have noticed spurious whitespace in at least two other treebanks.

2.3. Other delimiting characters

Sometimes, characters other than whitespace are used to separate one field from another in a treebank. Typically colons or semicolons, single or double quotation marks and round, square, curly or angle brackets fill this role. We have encountered four treebanks where for various sentences one of these characters was missing, doubled, misplaced, or replaced by another character.

These errors are particularly serious if they occur with XML delimiters, such as the quotation marks around attribute values or the angle brackets around XML tags. In one treebank, several files were not valid XML due to this kind of error.

2.4. Character encoding

We have encountered one treebank in which some files were encoded in UTF-8 (Unicode) and others in the Windows-1254 encoding.

2.5. Encoding of special characters

In spite of Unicode, which can encode practically all languages, language group-specific encodings remain in use. In fact, only one treebank we encountered uses

¹Many thanks to the other organizers: Erwin Marsi, Amit Dubey and Yuval Krymolowski.

²<http://ilps.science.uva.nl/erikt/signll/conll/>

Unicode (Hajič et al., 2004). Sometimes, an encoding (e.g. ASCII or one from the ISO-8859 family) is used for a treebank because it is the standard encoding for the treebank’s language, but the texts on which the treebank is based originally contained a few characters that are not covered by the encoding. A common solution is then to encode these characters as named character entities. For example, the Prague Dependency Treebank (Böhmová et al., 2003) uses `à` for à, the British component of the International Corpus of English (ICE-GB) (Greenbaum, 1996) uses `°ree;` for ° and SUSANNE (Sampson, 1995) uses `<deg>` for the same symbol. In ICE, we have encountered some inconsistencies in these entity names, e.g. both `&much-less-than;` and `&much-smaller-than;` and both `Β` and `&BETA;` are used. In addition, some special characters that should have been encoded were not.

Certain characters have a special meaning in most annotation schemes (see Section 2.3.). Therefore, they should be encoded or escaped when they are meant to represent the original character. In XML, either single or double quotation marks must enclose attribute values.³ If the attribute value is meant to contain the same quotation mark as is used for the enclosure, it has to be encoded as `'` or `"`; respectively.⁴ In addition, `&` has to be encoded as `&`.⁵ We have encountered two treebanks that failed to do that and therefore contained files that were not valid XML.

Actually, the need to encode quotation marks can be avoided by putting information such as the stem or lemma in an element of its own instead of some element’s attribute.⁶ However, this annotation style was not chosen in any of the four XML-encoded treebank which we encountered that contain lemma information⁷ (although it is used in the SGML version of the Prague Dependency Treebank). Three treebanks even encode the word forms as attributes in their XML versions.

2.6. Presence and order of all fields

We have encountered two treebanks where the POS tag for a word was missing, another one where the lemma information was missing in part of the treebank, a fourth one where one word token had two conflicting PoS attributes and a fifth one where one lemma was empty, four constituent labels were missing and one constituent had two different function labels. Two of these treebanks also had at least one case where pieces of information (such as lemma, POS and other features) were in a different order than in the rest of the treebank.

³`<token lemma="O'Neill"> O'Neill </token>`

⁴`<token form='O'Neill'> O'Neill
</token>`

⁵See the W3C Recommendation (<http://www.w3.org/>) on XML 1.1, Section 2.4 “Character Data and Markup”

⁶`<token> <lemma> O'Neill </lemma> <form>
O'Neill </form> </token>`

⁷Possibly because it was felt that only the original word forms are “data” and everything else is meta-data.

2.7. Typos in labels

We have encountered five treebanks that had typos in labels, i.e. in the names of POS, constituents, grammatical functions or additional features. There was also at least one case of the lemma being for a completely different word than the token itself and one treebank where a few XML labels had inconsistent case (`<W>` ended by `</w>` and vice versa).

2.8. Typos in structure

Tree structure can be expressed in various ways: for example, phrase structure by indentation or paired brackets or tags, dependency structure by a dependency tree with indexes on the leaves indicating linear order or by a linearly ordered list where each token has a field containing the index of its head. In all these cases, typos in the structure can lead to incorrect annotation. If the result is incorrect attachment, it is very hard to detect automatically. However, we have encountered four types of problems that can be spotted automatically. One XML-encoded phrase structure treebank contained more than 40 cases of XML tags that were not properly nested. One dependency treebank contained some tokens for which the index of the head was larger than the highest index in the sentence. Two dependency treebanks contained sentences with dependency cycles, i.e. where either a token directly links to itself or it links to another token that links to another token, etc., that links back to the first. One treebank encoded phrase structure but allowed for discontinuous constituents. The beginning and end of a discontinuity in a constituent xp is marked by a mirrored pair of symbols $xp-$ and $-xp$. We have encountered cases where there were beginnings without ends and vice versa and at least one case where the depth of the beginning and end was different (although it must be identical). In theory, the same problem can occur with normal phrase structure brackets, although we have not encountered that.

2.9. Potential errors in the linguistic annotation

It is rare that one notices errors in the linguistic annotation for a language one does not speak. However, sometimes a general linguistic understanding of a treebank’s annotation scheme is enough to at least suspect that something is wrong (and consequently report it to the treebank’s authors). For example, one treebank has separate labels for finite, non-finite and averbal (sub)clauses. Given these labels, we suspect an error when we encounter, for example, a constituent without a verb being labelled a finite clause or a constituent with a verb labelled an averbal clause. Sometimes a dependency treebank annotation scheme contains a designated label for the root token of the whole sentence or clause. For example, in the Metu-Sabancı treebank (Ofłazer et al., 2003; Atalay et al., 2003), this is the SENTENCE label. So we suspect an error when we encounter a sentence that does not contain this label.

3. Preventing and detecting problems

In the previous section, we listed many types of problems that we noticed in various treebanks. In this section, we propose ways to prevent or detect these problems.

3.1. Explicit documentation

We think that the first step towards ensuring that a treebank does not exhibit any of the problems discussed in Sections 2.1. to 2.7. is to explicitly document the conventions used for a treebank. This will raise awareness of these conventions in everybody working on or with the treebank. In particular, machine-readable lists of all special character encodings and labels used in the treebank would be very helpful. Creating such lists encourages treebank providers to check for mistakes, and having such lists readily available encourages script writers to actually check whether the data adheres to it. The majority of the treebanks do not provide such lists in easily machine-readable form (although sometimes they can be copy-pasted from web pages or PDF files).

3.2. Format checker

An explicit documentation of conventions can be used as a specification for an automatic format checker, i.e. software that reads in treebank files and flags cases that violate the specification. Such software could automatically detect the problems described in Sections 2.1. to 2.8. Some treebank projects use tools, such as Annotate (Plaehn, 2002), for treebank creation/editing that directly enforce the format, for example by providing a pull-down menu of labels instead of having annotators type in labels by hand. However, creating such a tool from scratch or even adapting an existing one to a different annotation scheme can be too big an overhead for a treebank project. The next best thing then is to write a format checker, and use it at regular intervals, ideally after each editing session. Once explicit documentation is available, writing such a format checker should be relatively easy for somebody with experience in scripting languages such as Perl or Python. If the treebank authors do not have this experience themselves, they should encourage users to write and submit such software, as it is in everybody's interest that it is used.

Some people might think that some treebank formats, such as XML, are inherently less susceptible to format errors. In our experience, however, this is not true. We have encountered at least two treebanks that use the XML format but apparently have been created or modified with a simple text editor and without validating the XML afterwards (see Sections 2.3. and 2.5.). In fact, only three out of seven XML-encoded treebanks came with a DTD or a reference to an XML Schema. We have also seen several scripts for converting treebanks to or from XML that parse or construct XML "by hand", i.e. through regular expressions and print statements instead of by using available XML library functions. This method is error-prone and should be avoided. In general we think that writing a format checker for a non-XML format is not more difficult than writing a DTD/Schema for an XML-based format. Note also that some restrictions, e.g. on cycles cannot be expressed in a DTD/Schema.

In general, we have not found one format better than another. We have, however, noticed two practices which we would discourage as they make the introduction of errors easier and their automatic detection harder. The first one is using XML attribute values with internal structure, e.g.

attribute values (encoded as strings) that are lists or sets of more atomic values, such as morphological feature values. Such an annotation style means that annotators have to deal with two different ways to encode structure, one that uses XML tags and one that uses for example brackets. It also means that these attributes cannot easily be validated through a DTD/Schema. The second practice that we would like to discourage is to keep different levels of annotation in separate files, e.g. one file with the part-of-speech annotation and another with the syntactic annotation of the same text. This approach has several disadvantages:

- Annotators of one level do not see the other level, so there is a higher chance of inconsistent annotation and a missed chance for detecting errors in the lower levels (e.g. the Penn treebank (Marcus et al., 1994) contains some VPs whose only verb is tagged as a noun).
- Some information, such as the words themselves, has to be repeated. This can lead to inconsistencies between the two versions if any corrections (e.g. of typos or tokenization errors) or changes (e.g. to the multiword policy) are carried out in only one version.
- Format checkers and especially conversion scripts would have to read in and parse more than one file at a time and establish the correspondence between tokens before being able to check or convert the content. This makes these tools more complicated and therefore less likely to be developed and used.

We have encountered at least one treebank that used the separate-file approach and suffered from not being format checked.

3.3. Conversion by head table

The errors in clause labelling described in Section 2.9. were detected during the conversion from the original phrase structure to the shared task dependency format. Jelinek et al. (1994) introduced the idea of a head table to automatically determine the head child of each constituent in a phrase structure treebank and Magerman (1995) used the first version for the Penn Treebank (Marcus et al., 1994). Collins (1996) slightly modified that table and used it to convert the Penn Treebank phrase structures into a collection of bilexical dependencies. Yamada and Matsumoto (2003) and Nivre and Scholz (2004) built on that idea and used a slightly modified version of Collins' head table to convert the Penn Treebank into an unlabelled or labelled dependency treebank respectively, on which a dependency parser can be trained and tested. A similar approach was used to convert the Alpino (van der Beek et al., 2002), BulTreebank (Simov et al., 2005; Marinov and Nivre, 2005), Bosque (Afonso et al., 2002), Cast3LB (Civit Torruella and Martí Antonín, 2002), Sinica (Chen et al., 2003), Talbanken05 (Nilsson et al., 2005), TIGER (Brants et al., 2002) and Japanese Verbmobil (Kawata and Bartels, 2000) treebanks to labelled dependency treebanks (or partially labelled in the case of Cast3LB).

If one looks at Magerman's and especially Collins' head table (Collins, 1999, p.240), one notices a number of linguistically implausible potential head children, e.g. ADJP, NN,

NNS⁸ or NP as head children of VP. For anyone who has worked with the Penn Treebank, it is clear that these try to compensate for annotation errors in the treebank. Although this makes sense for the researcher only interested in converting a messy treebank as well as possible, we propose to actually use the conversion by head table as an instrument of quality control. This can be achieved by three additions to the head table format.

Firstly, a flag for each element in a rule stating whether this is a linguistically sound or rather a “heuristic” potential head child. Whenever a “heuristic” rule part has to be used, the head finding algorithm should output a warning about this fact. Secondly, the direction is not specified per line (parent constituent) but per element (head child). This was implicitly already needed for Collins (1999)’s special rules for NPs and explicitly realized in Bikel (2002)’s reimplementation of Collins’ parser. Thirdly, in addition to the directions “left” and “right” that state that the head child is the leftmost, respectively rightmost, matching child, we propose a requirement of “exactly one”.⁹ If for example one thinks that an adjective is a potential head child of an ADJP, and that at most one adjective should occur as a direct child of an ADJP, one would write a rule ADJP: "only" A. The algorithm that applies the head table should then output a warning whenever it tries to apply an “only” rule in a context where there are several matching elements. “Only” rules will be especially frequent for treebanks such as Alpino, Bosque, Sinica, Talbanken05, TIGER and Verbomobil, which explicitly mark heads, as the linguistic definition entails that there should be only one head (although multi-words can sometimes be an exception). The proposed method for detecting errors, which involves linguistic understanding, can be complemented by fully automatic methods such as Ule and Simov (2004) and Dickinson and Meurers (2005).

4. Handling problems

Magerman’s original head table dates from 1995 and the much-cited work by Collins that also uses it dates from 1996 and 1997. There has been a new release of the Penn Treebank since then (Treebank-3, 1999). However, many of the errors that necessitated the “heuristic” head rules are still in the treebank, see e.g. (Dickinson and Meurers, 2005). There are two possible explanations for this: Either the treebank users did not tell the treebank authors about these errors¹⁰ or the treebank authors did not do anything with this information.

Whenever we or one of the shared task participants¹¹ noticed errors in the shared task data that were due to errors in the original treebank (and not to conversion bugs), we reported these back to the treebank providers, and we are very pleased to report that, whenever feasible, they were

⁸NN and NNS are the Penn Treebank POS tags for singular and plural nouns, respectively.

⁹Based on an idea by Montserrat Civit (p.c.), although with a slightly different interpretation.

¹⁰Note that Michael Collins was a PhD student at the University of Pennsylvania itself at that time.

¹¹Thanks to Ryan McDonald, Svetoslav Marinov and Masayuki Asahara for reporting bugs.

corrected, sometimes within days. As a consequence, some of the errors reported in this paper are no longer present in the respective treebanks.

4.1. The challenges

We think that this feedback loop is an important part of quality control and should be encouraged explicitly. This can be done by clearly stating in the treebank’s documentation that bug reports are welcome and by providing a contact address for reporting them. Although a quick fix of reported problems is the best encouragement for users actively working with the treebank to report further problems, should they find any, this might not always be feasible. In addition, treebank providers might be reluctant to officially release new versions of a treebank at very short intervals or for very minor changes. Also, funding for the project might have ended, and the original authors might have moved on to other jobs. In that case, even if users notice errors and fix them in their own copies of the treebank, there is currently no easy way for future users to profit from those fixes, as license restrictions often prohibit one user from passing on modified versions to another. Finally, when treebanks are used to train and test systems such as taggers and parsers, and when one wants to be able to compare results by different systems at different times, it is vital that one compares against exactly the same version of the treebank.

4.2. Patching treebanks

We think that we can learn from software developers, especially the open source community, how to achieve these goals and overcome these problems. We introduce the concept of “patches” to treebanks. In essence, treebanks are text files, just as source code is. After a treebank author or user has made changes to a treebank in order to fix reported or noticed problems, the Unix `diff` utility allows them to list all changes in a very concise manner. If produced with the options `-c` or `-C`, the output of `diff` can be used as the input of the Unix `patch` utility to apply the same changes to somebody else’s copy of the original treebank. This means that

- if users think they know how to fix a problem, they can make fixes to their copy and then have a very easy and foolproof way to report back to the treebank authors how they propose to change the treebank.
- treebank authors then simply have to decide whether to accept or reject the proposed patch; in either case they can simply post it to the treebank’s web site with an explanation. This directly makes accepted patches available to other users without the need for a new release of the treebank. Also, authors can automatically apply patches accepted since the last release for the next release, thereby keeping their own effort to a minimum.
- if a treebank is not maintained any longer by the license holders, users can probably share patches among themselves without violating the treebank’s license, as patches only contain fragments of the treebank texts and are useless without the original treebank. This would allow even the Penn Treebank to be

patched. Obviously there is a risk of diverging or disputed fixes, as noted in Blaheta (2002), and there is also the problem of new users not knowing about or not having access to older fixes.¹²

4.3. Versioning

Out of the twenty-one treebanks we studied, four used numbers for major releases/versions only (e.g. TIGER Version 1 and 2), nine used two-part numbers for the major and minor release (e.g. Bosque 7.3) and eight did not use any apparent version numbers. We propose that all treebanks follow a versioning scheme similar to software, with major and minor releases and numbered patches. This will allow researchers to state precisely which version of a treebank their reported results pertain to. Ideally, all old versions should be kept available, so that if some researchers wants to replicate or compare to somebody else's results years later, they can still get the appropriate version. They can then state something like "applying our new method to the same treebank version as X, we get $y\%$ while X got only $z\%$; applying our method to the latest version ($i.j.k$), we get $u\%$ ". As treebanks can be big, treebank authors might decide to keep only major releases and allow users to reconstruct in-between versions by applying a number of patches to the preceding major release.

Obviously the boundary between a major and a minor release or between a minor release and a patch is a fuzzy one but the term "release" seems to suggest some important improvement has been made. Therefore treebank providers are probably reluctant to make a new "release" if the changes "only" fix problems such as the ones discussed in Sections 2.5. to 2.7. and prefer to wait for the next release, which might be years in the future. It is for the quick fixes to these problems in particular that we advocate the use of patches.

Finally, we can only recommend the use of versioning software such as CVS (Concurrent Versions System)¹³ or Subversion¹⁴, both of which are available for free, to help keep track of versions.

5. Summary

We have described the quality problems we encountered during our work with a large number of treebanks. We propose preventing and detecting these problems through the explicit documentation of format conventions, in particular machine-readable lists of labels, through the use of format checkers and, for phrase-structure treebanks, through additions to the head table mechanism. We also plead for a structured approach to handling reported problems and propose to adopt the software concepts of patching and versioning. We hope that this paper will help to improve existing and future treebanks.

Acknowledgements

Many thanks to the treebanks providers who kindly allowed the use of their treebanks for the shared task, answered

questions about the annotation and fixed errors, to the other organizers and some treebank providers who converted the treebanks and to the shared task participants who asked useful questions and noticed bugs. Without the help of all these people the work on which this paper is based would not have been possible.

6. References

- I. Aduriz, M. Aranzabe, J. Arriola, A. Atutxa, A. Daz de Ilarraza, A. Garmendia, and M. Oronoz. 2003. Construction of a Basque dependency treebank. In *Proc. of the Second Workshop on Treebanks and Linguistic Theories (TLT)*.
- S. Afonso, E. Bick, R. Haber, and D. Santos. 2002. "Floresta sintá(c)tica": a treebank for Portuguese. In *Proc. of the Third Intern. Conf. on Language Resources and Evaluation (LREC)*. ELRA.
- N. Atalay, K. Oflazer, and B. Say. 2003. The annotation process in the Turkish treebank. In *Proc. of the 4th Intern. Workshop on Linguistically Interpreted Corpora (LINC)*.
- D. Bikel. 2002. Design of a multi-lingual, parallel-processing statistical parsing engine. In *Proc. of the Human Language Technology Conf. (HLT)*.
- D. Blaheta. 2002. Handling noisy training and testing data. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, pages 111–116.
- A. Böhmová, J. Hajič, E. Hajičová, and B. Hladká. 2003. The PDT: a 3-level annotation scenario. In A. Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*.
- S. Brants, S. Dipper, S. Hansen, W. Lezius, and G. Smith. 2002. The TIGER treebank. In *Proc. of the First Workshop on Treebanks and Linguistic Theories (TLT)*.
- S. Buchholz, E. Marsi, A. Dubey, and Y. Krymolowski. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. of the Tenth Conf. on Computational Natural Language Learning (CoNLL-X)*. SIGNLL. To appear.
- K. Chen, C. Luo, M. Chang, F. Chen, C. Chen, C. Huang, and Z. Gao. 2003. Sinica treebank: Design criteria, representational issues and implementation. In A. Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*.
- M. Civit Torruella and M^a A. Martí Antonín. 2002. Design principles for a Spanish treebank. In *Proc. of the First Workshop on Treebanks and Linguistic Theories (TLT)*.
- M. Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proc. of the 34th Annual Meeting of the ACL*.
- M. Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proc. of the 35th Annual Meeting of the ACL*.
- M. Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.
- M. Dickinson and W. D. Meurers. 2005. Prune diseased branches to get healthy trees! In *Proc. of the Fourth Workshop on Treebanks and Linguistic Theories (TLT)*, pages 41–52.
- S. Džeroski, T. Erjavec, N. Ledinek, P. Pajas, Z. Žabokrtsky, and A. Žele. 2006. Towards a Slovene

¹²For example, the URL mentioned in Blaheta (2002) for his list of corrections to the Penn Treebank does not exist anymore.

¹³See e.g. <http://www.nongnu.org/cvs/cvs.html>

¹⁴See <http://subversion.tigris.org/>

- dependency treebank. In *Proc. of the Fifth Intern. Conf. on Language Resources and Evaluation (LREC)*.
- S. Greenbaum, editor. 1996. *Comparing English Worldwide: The International Corpus of English*. Clarendon Press, Oxford.
- J. Hajič, O. Smrž, P. Zemánek, J. Šnaidauf, and E. Beška. 2004. Prague Arabic dependency treebank: Development in data and tools. In *Proc. of the NEMLAR Intern. Conf. on Arabic Language Resources and Tools*.
- E. Jelinek, J. Lafferty, D. Magerman, R. Mercer, A. Ratnaparkhi, and S. Roukos. 1994. Decision tree parsing using a hidden derivation model. In *Proc. of the Workshop on Human Language Technology*.
- Y. Kawata and J. Bartels. 2000. Stylebook for the Japanese treebank in VERBMOBIL. Verbmobil-Report 240, Seminar für Sprachwissenschaft, Universität Tübingen.
- M. T. Kromann. 2003. The Danish dependency treebank and the underlying linguistic theory. In *Proc. of the Second Workshop on Treebanks and Linguistic Theories (TLT)*.
- D. Magerman. 1995. Statistical decision-tree models for parsing. In *Proc. of the 33rd Annual Meeting of the ACL*.
- M. Marcus, G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. 1994. The Penn treebank: Annotating predicate argument structure. In *Proc. of the Workshop on Human Language Technology*.
- S. Marinov and J. Nivre. 2005. A data-driven dependency parser for Bulgarian. In *Proc. of the Fourth Workshop on Treebanks and Linguistic Theories (TLT)*, pages 89–100.
- J. Nilsson, J. Hall, and J. Nivre. 2005. MAMBA meets TIGER: Reconstructing a Swedish treebank from antiquity. In *Proc. of the NODALIDA Special Session on Treebanks*.
- J. Nivre and M. Scholz. 2004. Deterministic dependency parsing of English text. In *Proc. of the 20th Intern. Conf. on Computational Linguistics (COLING)*.
- K. Oflazer, B. Say, D. Zeynep Hakkani-Tür, and G. Tür. 2003. Building a Turkish treebank. In A. Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*.
- O. Plaehn. 2002. Annotate Bedienungsanleitung. NEGRA project report, Universität des Saarlandes, Germany.
- G. Sampson. 1995. *English for the Computer: The SUSANNE Corpus and analytic scheme*. Clarendon Press.
- K. Sima'an, A. Itai, Y. Winter, A. Altman, and N. Nativ. 2001. Building a tree-bank of modern Hebrew text. In B. Daille and L. Romary, editors, *Journal Traitement Automatique des Langues (t.a.l.) — Special Issue on Natural Language Processing and Corpus Linguistics*.
- K. Simov, P. Osenova, A. Simov, and M. Kouylekov. 2005. Design and implementation of the Bulgarian HPSG-based treebank. In *Journal of Research on Language and Computation – Special Issue*, pages 495–522. Kluwer Academic Publishers.
- T. Ule and K. Simov. 2004. Unexpected productions may well be errors. In *Proc. of the Fourth Intern. Conf. on Language Resources and Evaluation (LREC)*, pages 1795–1798.
- L. van der Beek, G. Bouma, R. Malouf, and G. van Noord. 2002. The Alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN)*.
- H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. of the 8th Intern. Workshop on Parsing Technologies (IWPT)*.