22

Reports from the ETAP project
Editor: Lars Borin

# Segmenting and Tagging Parallel Corpora

Papers by:

Camilla Bengtsson
Lars Borin
Henrik Oxhammar

research report
etap-rr-05
2000

# Segmenting and tagging parallel corpora

Papers by

Camilla Bengtsson
Lars Borin
Henrik Oxhammar

WP CL&LE 22

# The ETAP project • Research reports

ETAP is short for *Etablering och annotering av parallellkorpus för igenkänning av översättningsekvivalenter* ("Creating and annotating a parallel corpus for the recognition of translation equivalents").

The basic aim of the project is to develop a computerized multilingual translation corpus, made up of Swedish source text representing different styles and domains, together with its translations into several languages, which can be used in bilingual lexicographic work and in methodological studies directed towards the development and evaluation of corpus formats and computational tools for the automatic recognition and extraction of translation equivalents from text.

The project is part of the research programme *Översättning och tolkning som språk- och kulturmöte* ("Translation and Interpreting—a Meeting between Languages and Cultures"), financed by the Bank of Sweden Tercentenary Foundation. This research programme

"...started in 1996. It involves a great variation of research topics within the domain of translation and interpreting and has an overall aim of seeing translation and interpreting as activities that are related not only to linguistic and textual aspects but to cultural, historical, social and communicative phenomena as well. [It] is a result of a collaboration between two big and well-known Swedish universities, Stockholm University and Uppsala University." (From the WWW homepage of the programme: <http://www.translation.su.se/abstract.html>)

WWW: http://stp.ling.uu.se/etap/

**ETAP research reports 2000:**

| | |
|---|---|
| etap-rr-04<br>(WP CL&LE 21) | *Seeing double: using parallel corpora for linguistic research*<br>Papers by Borin, Olsson, Prütz |
| etap-rr-05<br>(WP CL&LE 22) | *Segmenting and tagging parallel corpora*<br>Papers by Bengtsson, Borin, Oxhammar |
| etap-rr-06<br>(WP CL&LE 23) | *ETAP project status report December 2000*<br>Lars Borin, with contributions by others |

# Contents of this volume

# Sentence splitting and SGML tagging

Henrik Oxhammar
Lars Borin
henriko@stp.ling.uu.se, Lars.Borin@ling.uu.se

**Abstract**

In this paper, a SGML tagger is described that is language independent and uses pattern matching for recognizing sentence boundaries.

This markup uses regular expressions for recognizing overall abbreviation patterns and date expressions, improving the sentence splitting function. A lexicon may or may not be used to further improve the results.

With this method a higher accuracy rate was achieved, in comparison with an earlier SGML markup program used in the department.

## 1 Introduction

In this paper, we report on work carried out as part of the ETAP project, in the Department of Linguistics, Uppsala University in the spring of 1999. The first author would like to thank Erik Mats for his help and comments.

The purpose of the work described here was to make a language independent, SGML sentence markup program. It was intended as an improvement on a previous program used in the ETAP project for the same purposes.

Marking sentence boundaries is not a trivial task (Grefenstette and Tapanainen 1994). Sentence punctuation marks like the exclamation point and question mark seldom cause any trouble. In contrast, the most common sentence delimiting punctuation mark, the full stop, is highly ambiguous. It can mark the end of a sentence, an abbreviation or both, i.e. an

abbreviation at the end of a sentence. It also has uses inside numerical expressions, such as the decimal point in English, the point used for grouping the figures in large numbers in Swedish (corresponding to a comma in English), or the point separating hours and minutes in time expressions in Swedish (corresponding to a colon in English). In order to distinguish these cases, a regular expression grammar would have to be used. But, as the reader probably understands, not all cases can be recognized. Take for example this German sentence (from the ETAP EU subcorpus; see Borin 2000):

(1) Ziel 3: Vereinbarkeit von Beruf und Haushalt bzw. Familie
fuer Frauen und Maenner

In this case there is no way for a computer to determine if the period marks a sentence boundary or an abbreviation (If not some kind of language understanding mechanism is used). A simpler solution that is not waterproof is to assume that abbreviations do not occur at the end of sentences and to have a lexicon listing the abbreviations.

However, there are some patterns that can be easily recognized and improve the result of marking sentence boundaries.

# 2 Background

## 2.1 Introduction

The purpose of this work was to either modify or create a new SGML markup program that would be used for the sentence alignment of text. The previous program used was a shell script (Tjong Kim Sang 1999), which did not perform all that well. It could not recognize obvious abbrieviations and dates. Since the markup also should be language independent, it was decided that a new program would be written.

Text that the program was to be tested on included Swedish, Spanish, German, Polish, Serbian-Bosnian-Croatian and English texts from the ETAP Scania subcorpus and the IVT (Invandrartidningen) subcorpus (see Borin 2000).

## 2.2 Sentence boundaries

Some of the first cases that had to be dealt with separately were dates and numeric expressions:

(2) Geschehen zu Bruessel am 24. April 1996.
(3) 1.2. Der Ausschuss begruesst ...

These were wrongly marked as (sentences are delimited by the SGML tags <s> ...</s>):

```
<s>
Geschehen zu Bruessel am 24.
</s>
<s>
April 1996.
</s>
```

and:

```
<s>
1.2.
</s>
<s>
Der Ausschuss begruesst ...
</s>
```

Both these should be marked as one sentence only.
These two types were very frequent in the texts. Recognizing these (and other) patterns would improve the sentence markup a great deal.

The existing program recognized a sentence as a punctuation mark followed by a whitespace. (Tjong Kim Sang 1999: 7-8). Another approach was decided upon. A larger context was to be used to recognize exceptions that should not be seen as sentence boundaries. By recognizing these patterns the markup would be improved.

In order to do this, these exceptions had to be established. So, the next thing to do was to gather as much knowledge as possible about tokens or sequences of tokens that should not be marked as sentence boundaries.

It was now decided that 'Flex' would be used. Flex is a stream scanner that recognizes patterns and executes an action written in C code.

## 2.3 Non-sentence boundaries

### 2.3.1 Types

In this approach, the criteria for a sentence boundary is a sentence punctuation mark (.!?) followed by whitespace. Semicolon was also decided to mark sentence boundaries since otherwise some sentences would become very long, which would make the subsequent sentence alignment step harder.

There were mainly five types of exceptions discovered:

- Date expressions
  '24. 12. 1994', '24. December 1994'
- Sequences of either upper or lower case
  i.st.f. (SWE), z.B. (GER)
- Single upper or lower case
  p. (SPA), V. (ENG)
- Sections and subsections
  '2. Introduction' and '1.2.3. Once upon ...'

A criterion common for almost all patterns was that there had to be a lowercase letter immediately after the pattern. By using this as a criterion, it would be certain that there was no sentence boundary. Only one pattern did not obey this criterion, namely the single upper or lower case letter. These were very frequent in the texts so it was decided to have them as a rule rather than forcing the user to explicitly stating them in the lexicon.

# 3 Implementation

## 3.1 Flex code

As already mentioned, the program was written in *Flex*. There are several advantages using Flex. First, it is a character scanner. Therefore this had not to be built separately. Second, unlike other UNIX-type stream editors, Flex can in its regular expression rules refer to more than one line (it is able to recognize newline and carriage return) thus providing a larger context which is very important in these cases. Another advantage is the speed. It is a very fast scanner.

To simplify, some general definitions are used, which are used in the rules to refer to classes of characters. These include characters that should be regarded as word delimiters, lower- and uppercase letters, and numbers.

A lexicon is also available. Here the user can state e.g. all patterns that should be regarded as abbreviations. In a later version this lexicon will be in a separate file. Example of lexicon:

```
LEXICON "ABl."|"Nr."|"S."|"np."
```

Rules where used for recognizing those cases that were regarded as abbrieviations, sentence boundries, headers and other patterns.

Currently there are 19 rules. Ten of these are rules matching patterns that are regarded as abbreviations. Five match patterns that should be regarded as sentence boundaries. One rule is for matching abbreviations listed in the lexicon. One rule is for recognizing paragraphs, and two other match everyting else.

Below each rule is explained in more detail.

- `{LEXICON}`

  This rule matches anything in the lexicon.

- `({WD}+)?{CHARS}+{WD}?+\.{WD}+{LC}+`

  This is the general case. It states that any sequence of {CHARS} followed by a period before any lowercase letter is an abbreviation. This is used to pick up any case of abbreviation that is not matched by the other rules. Tokens like 'täyd.' (Finnish) and 'Temp.' (English) will be considered in the given context as abbreviations.

- `({WD}+)?(\/)?{NUM}{1,2}\.{WD}({UC}{LC}+){WD}{NUM}{4}`

  There were several ways found to express dates in the different texts. This rule matches patterns like '24. Februar 1988'. As the reader can see, this is a language specific rule for German. The reason for using this language dependent rule was that it was very frequent in some of the texts. If the rule produced to many false hits, it was decided that it would be removed later. The rule is as specific as possible, only accepting one or two figures first (days) and exactly four figures last (year). This would limit the number of matches.

- `{WD}+{NUM}{1,2}\.{WD}{NUM}{1,2}\.{WD}{NUM}{4}`

  This rule recognizes another type of date expression. These were mostly found in the Swedish and German texts. Date expressions like '17. 12. 1973' will be recognized.

- `{WD}+({UC}+|{LC}+)\.(({WD}?{UC}+|{LC}+)\.)+{WD}+{LC}+`

  This rule matches all cases of any letter followed by a period, two or more times. These patterns are very frequent in many of the languages. Swedish has e.g. 'i.st.f.' (in stead of), English has 'i.e.' and German has 'z. B.' (for example).

- `{WD}+({LC}|{UC})\.`

  With this rule a single upper- or lowercase letter followed by a period is matched. This rule differ from the other rules in that it does not require a lowercase letter after the period. This is because it is for recognizing patterns like 'S. 34' in German and 'E. P. Garcia' in Spanish. It should be safe to assume that sentences do not end in a single letter.

- `{WD}+{CHARS}+(\.)?\/{CHARS}+\.{WD}+{LC}+`

  One of the subcorpora used is the ETAP Scania subcorpus. This corpus is a collection of truck manuals from the Swedish truck manufacturing company Scania, containing highly technical text material. This rule matches patterns like 'r/min.' (Swedish) and 'acopl./desac.' (Spanish) and 'km/h.' (English).

- `[\.]{2,}({WD}+)?{LC}+`

  Although not very frequent, these cases had to be recognized and treated as abbreviations. These are cases like this 'Gleichbehandlung ... fuer'. The rule matches two or more periods followed by a lowercase letter. These types should however occur in more languages than German.

- `^{NUM}{1,2}\.({NUM}+(\.)?)+{WS}+{CHARS}+`

  With this rule subsections are recognized. The rule matches one or two numbers at the beginning of a line followed by one or more of a sequence of one number followed by a (optional) period followed by any character. These expressions were treated as abbreviations although they actually are not. It felt more intuitively right that the numeric expression should occur with the rest of the sentence and not as its own sentence.

- `^{NUM}{1,}\.({WS}+{CHARS}+){3,}`

  The rule matches one or more numbers followed by a period followed by three or more words. I might seem strange to treat these cases as abbreviations. The key is that this only matches the beginning of lines. It had to be stated due to another rule (see below) and distinguishes the patters from each other. This type of pattern was very frequent. Applying this rule improved the segmentation a lot. For example: '9. Je$eli nie dostaniesz pracy to nauczysz' (Polish) would be marked as one sentence.

- `^{NUM}{1,}\.({WS}{CHARS}+){1,2}(\.)?`

  Unlike subsections, headers would be treated as sentences. As seen headers are treated as one or more numbers at the beginning of the line followed by a period followed by one or two words. This was decided in order to distinguish it from the previous rule. This may not work for all texts, but worked fine with the ETAP texts.

- `(\.)?({WS}+)?\n[\n\r]+`

  Paragraphs also had to be tagged. Therefore they had to be recognized. The rule matches an optional period and whitespace, followed by two or more newlines or carrige returns (Windows).

- `^[A-Z]{1,4}\.{WD}`

  This is another way to write headers and subsections in the texts. In this case Latin characters followed by a period are used. Like subsections these were treated as abbreviations. For example 'III. Rådet gav ...' would be matched.

- `\!{WD}`

- `\?{WD}`

- `\.{WD}`

- `\;{WD}`

  The previous all declare sentence boundaries. The reason for {WD} is that there are cases when a period occurs for instance before a ' " '. The colon should be counted as belonging to that sentence and the sentence end should be after the colon. As previously mentioned, it was decided to treat the semicolon as a sentence separator, because some of the texts would otherwise have extremely long sentences.

- `.`

  This rule matches everything that is not matched by any other rule. See below for an explanation.

- `\n`

  This rule matches newlines. To make the printout look better, newlines are replaced by whitespace.

## 3.2 C code

The c code is rather simple. The algorithm is as follows:

```
- Read text to a buffer
- Check if any of the stored text match any of the rules
        - If pattern matched is regarded as sentence boundary
          - Accept as sentence (see below)
        - If a new paragraph is matched, accept text stored so far as a
          sentence and mark the paragraph, print and empty buffer.
- EOF
        - If buffer is not empty
          - Print the last text and mark it as a paragraph

- Function acceptAsSentence
        - Add sentence start tag to paragraph buffer
        - Concatenate current string with previous
        - Add sentence end tag to paragraph buffer
        - Flush the sentence buffer
```

The program assumes two things. That a text consists of at least one paragraph and at least one sentence.

The text is read into a buffer. Depending on what is matched, the stored text is regarded as a sentence or not. If what is matched is regarded as an abbreviation it is ignored. For each paragraph, a buffer is filled with SGML-marked sentences. It is not until a new paragraph is encountered that the results are printed.

## 3.3 Post processing

The program described here is a part of a program suite which produces aligned parallel texts. The texts to be aligned included the pagebreak marker. This pattern had to be recognized in order to handle them correctly. They should not be marked with either sentence or paragraph

markers. It was decided that these cases were easiest handled with a post processor.

The post processor works on the SGML text created by the SGML markup program. It was also written in Flex. It recognizes the pagebreak pattern and simply removes surrounding SGML tags.

# 4 How to use the program

A shell script is used to run the program. It relies on two other programs, the SGML markup program and the post processor.

The script takes two arguments: (1) the file which should be SGML marked and (2), an optional flag, stating whether the lexicon should be used or not. Example of a call:

```
./markup.sh sbkIVT.txt -l
```

A file with the same name as the input file with the suffix '.sgml' will be created with the marked text, as well as a '.log' file with data on the number of marked sentences and paragraphs.

# 5 Results

This sentence markup program performs better than the previous markup program used in the ETAP project, which was the goal. By adding more linguistic knowledge the segmentation results improved, albeit variously, depending on the type of text to be marked.

Another improvement is the the time used. Six texts of different sizes were tested. The previous program marked the text in a total time of 95.3 seconds or an average of 15.8 seconds/text. This program had a total time of 33.7 seconds or an average time of 5.6 seconds/text. As can be seen this program took about a third of the time used by the previous program used by the Department.

The difficulty with abbreviations is that they may well be at the end of a sentence (see below for a possible solution). A further difficulty is with languages like German which use capitalization for marking nouns, adding another ambiguity. The same goes for proper names for every language used here.

Abbreviations can (as already discussed) end sentences. Some abbreviations are more likely to do this than others, but such statistics are language-dependent and have therefore been avoided here. Instead, it was decided that it was better to mark a sentence end in those cases that could not be determined. In many cases this turned out to be the right decision. Since the marked texts should be aligned it is better to mark sentence boundaries more times, thus giving smaller units to align and analyse.

# 6 Improvements

There are several things that should be done in a later version of this program. The first thing is a separate lexicon. As it is now, the lexicon is incorporated in the program. This means that the user has to compile the program each time the lexicon in updated.

Another feature that would be good to have in the program, is to state cases regarded as abbreviations as definitions. This would make it possible to state things like

```
{CHARS}\.{WD}+({LC}+|{ABR}+).
```

# References


Borin, Lars 2000 (with contributions by others) ETAP project status report December 2000. In: Lars Borin (ed.), Working papers in Computational Linguistics and Language Engineering 23: Reports from the ETAP project. Department of Linguistics, Uppsala University.

Grefenstette, Gregory and Pasi Tapanainen 1994. What is a word? What is a sentence? Problems of tokenization. *3rd Conference on Computational Lexicography and Text Research. COMPLEX'94*. Budapest.

Tjong Kim Sang, Erik 1999. Converting the Scania Framemaker documents to TEI SGML. In: Anna Sågvall Hein (ed.), Working papers in Computational Linguistics and Language Engineering 18: Reports from the ETAP project: Converting, aligning and tagging for ETAP. Department of Linguistics, Uppsala University.

# Comparing and combining part-of-speech taggers for multilingual parallel corpora

Camilla Bengtsson
Lars Borin
Henrik Oxhammar
camilla@stp.ling.uu.se, Lars.Borin@ling.uu.se, henriko@stp.ling.uu.se

**Abstract**

We report on two experiments conducted with freely available off-the-shelf part-of-speech taggers for English and German texts in the ETAP corpus, in order to

(1) Assess the accuracy of the taggers when applied to 'new' text types;

(2) Investigate whether a combination of taggers could be made to perform better than a single tagger.

The results show that reports in the literature on the accuracy of taggers should be interpreted with some caution, as they are not always generalizable, but also that there is reason to be optimistic as to the possibility of combining taggers to enhance accuracy, at least for the languages in question.

## 1 Introduction

We report on research conducted during a period in the spring and summer of 1999. The work was done as part of the ETAP project in the Department of Linguistics, Uppsala University.

ETAP is an acronym for the (Swedish) project name, *Etablering och annotering av parallellkorpus för igenkänning av översättningsekvivalenter* ("Creating and annotating a parallel corpus for the recognition of translation equivalents").

The project is part of the Stockholm-Uppsala Research Programme "Translation and Interpreting - A Meeting between Languages and Cultures" financed by the Bank of Sweden Tercentenary Foundation(Riksbankens Jubileumsfond).

The goal of the project is to develop a computerized multilingual corpus to be used in bilingual lexicographic work, for the extraction of translation equivalents for use in machine translation applications, as well as for the development of methodology and computational tools for the automatic recognition and extraction of translation equivalents.

The languages involved are: Swedish, Dutch, English, Finnish, French, German, Italian, Polish, Serbian-Bosnian-Croatian and Spanish.

Texts worked on include texts from the ETAP Scania subcorpus and The Swedish Statement of Government Policy subcorpus (Regeringsförklaringen). See Borin 2000a for a more thorough description of the ETAP project and the ETAP corpus.

The present report is about the task of finding part-of-speech (POS) taggers for the ETAP languages and evaluating the taggers for their usefulness on the ETAP corpus material.

Each author has contributed to this report as follows:

Camilla Bengtsson searched for taggers for the ETAP corpus languages, evaluated the English taggers and tagger combinations, and wrote the parts of the report where this work is described;

Henrik Oxhammar implemented the program for comparing the output of several taggers applied to the same text. He wrote the description of the implementation;

Lars Borin has been responsible for the overall design of the experiments and is the author of the final version of the report. He also evaluated the German taggers and tagger combinations.

# 2 Background

In order to annotate the corpus with part-of-speech categories and morphosyntactic features you have to get hold of POS taggers that can perform this task automatically and reasonably well. It is an obvious fact that tagging the texts manually would take ages and developing and training our own automatic taggers for several languages is not within the brief of the ETAP project (although there has been some work in this direction with Swedish taggers; see Prütz 1999a, 1999b). Thus what we need to do is to find already existing (public domain) taggers and evalutate them to see which ones are most suitable for the texts in the ETAP Project. Work has

been done to find POS taggers mainly for English, Spanish and Swedish, but along the way we have also come across possible taggers for French, Italian and Dutch. Evaluation of German taggers is described separately below.

There is an often-told anecdote about a night walker chancing upon an obviously inebriated man crawling on all fours under a street lamp, apparently searching for something on the ground. When asked, the man explains that he has lost his house keys, and cannot get into his home until he finds them again, whereupon the first man offers to help him in his search, and he, too, gets down on his knees. After a while, when it becomes apparent that there are no keys on the ground, nor any cracks or holes into which they could have fallen, the first man asks the drunk whether he is certain that he lost his keys exactly in the spot where they are looking. "No, I'm sure I lost them over there," the other man replies, and points into the surrounding darkness. "So why in the world are you looking for them here, then, if you know that you didn't lose them here?" "Because the light's much better over here under the street lamp, of course."

The situation with tagger evaluation—especially in languages other than English—is sometimes very similar to that in the anecdote; there is a street light—one tagged corpus, which everybody uses for tagger training and evaluation—and a lot of dark terrain—the text material for which the tagger is intended to be used, and not very much is known about the relationship between the two.

There is nothing particularly surprising about this. The tagging (manual or semi-automatic, i.e., automatic with manual proofreding) of a corpus is a vast undertaking. The Swedish Stockholm Umeå Corpus (SUC) is a tagged one-million word corpus—quite a modest size, by today's standards—balanced according to the same principles as the Brown and LOB corpora of English. SUC 1.0 appeared in 1996, after seven years of work, and a corrected ver. 2.0 is due to appear any day now.

Hence, we cannot take for granted that a POS tagger will perform with the same accuracy on new text material as on that on which it has been trained. The only way to find out is by empirical investigation. Exactly such empirical investigation is the topic of this report.

This section describes the work done so far on finding and evaluating POS taggers for the ETAP Project. The first section is about the search for taggers and the result of this search. The following sections deal with the testing and mapping of the English taggers and their tagsets.

## 2.1 Finding Taggers

The search for POS taggers has mainly been conducted by searching the Internet and by contacting people involved in various POS tagging projects. Since we needed to find well-performing taggers that were also in the public domain for academic research purposes our search possibilities were somewhat limited. In short there was not a big problem finding English taggers answering to these requirements, but it has proven difficult to find (free) Spanish taggers that can perform POS tagging of relatively raw texts and produce a disambiguated result, i.e. assigning exactly one tag to each token.

### 2.1.1 English

As stated before it was not very hard to find English taggers and we came up with three possible candidates: QTAG (Mason 1997), TreeTagger (Schmid 1994) and the AMALGAM Tagger (Atwell et al. 2000).[1]. The latter is a system (developed by the Natural Language Processing research group in the School of Computer Studies at Leeds University), that can be used via email[2] and from which it is possible to choose 8 different tagging schemes:

```
1. BROWN, 226 tags (Brown Corpus)
2. ICE,   205 tags (International Corpus of English)
3. LLC,   210 tags (Lundon-Lund Corpus)
4. LOB,   153 tags (Lancaster-Oslo/Bergen Corpus)
5. PARTS,  20 tags (UNIX parts)
6. POW,    66 tags (Polytechnic of Wales Corpus)
7. SEC,   150 tags (Spoken English Corpus)
8. UPENN,  45 tags (University of Pennsylvania Corpus)
```

TreeTagger uses the same tagset as the AMALGAM UPENN scheme above and QTAG uses the Birmingham-Lancaster Tagset which is a variant of the Brown/Penn tagsets and has 70 tags.

### 2.1.2 Spanish

It has been harder finding taggers for Spanish, possibly because of the fact that there are not many annotated corpora available to train the taggers on. The only Spanish tagger we have access to at the moment is CRATER (Sánchez León & Nieto Serrano 1995), which is the Spanish version of the

[1] See also <http://www.scs.leeds.ac.uk/amalgam/amalgam/amalghome.htm>

[2] amalgam-tagger@scs.leeds.ac.uk

Xerox Tagger, but in the near future we also expect to get a license for the Spanish version of the MBT Tagger (Daelemans et al. 1996).

### 2.1.3 Swedish

We currently have access to three Swedish taggers, i.e. the Brill taggers for Swedish developed by Prütz (1999a, 1999b), Lager (1999) and Ridings. The latter is available via the Internet as a part of speech tagging service[3]. We also expect to get a license for the Swedish version of the MBT Tagger (Daelemans et al. 1996).

### 2.1.4 Other

We have come across POS taggers for a couple of other ETAP languages as well, i.e. French (TreeTagger and Multext[4]), Italian (TreeTagger) and Dutch (MBT Tagger).

## 2.2 Testing

The next step after finding relevant taggers was to tag three text types of the ETAP project corpus with the different taggers and pick at random ten sentences from each of these tagged texts. This has so far only been done for English, but the same procedure will be applied to (first of all) Swedish, Spanish and French. The following sections all deal with the testing of the English taggers: the test texts, the ten randomly selected sentences and the results from comparing the tagger outputs.

The three (sample) texts of different types that have been tagged are a fragment of *Invandrartidningen* (the English version: 'News and Views'), *Regeringsförklaringen* ('Statement of Government Policy') and two *Scania* documents.

After tagging the above mentioned text types with the different English taggers: QTAG, TreeTagger and AMALGAM Email Service (8 different tagging schemes), ten sentences from each of the text types were picked out randomly from the TreeTagger texts and then the same sentences were picked out from the QTAG texts and from the resulting texts of the 8 different tagging schemes of the AMALGAM Tagger.

The ten tagged sentences of each text type and tagger were then checked and the percentage correct tags was counted. Taggers with a percentage of

[3] <http://www.gusd.holding.gu.se/>, choose *Ordklasstaggning*

[4] <ftp://issco-ftp.unige.ch/pub/multext/>

90 per cent or above correct tags were to be kept for further comparisons. The result from this process is as follows:

| *Invandrartidningen*: | |
|---|---|
| Tagger/tagset: | percentage correct: |
| Amalgam-sec | 97% |
| Amalgam-lob | 97% |
| Amalgam-upenn | 97% |
| TreeTagger | 96% |
| Amalgam-brown | 95% |
| Amalgam-ice | 94% |
| Amalgam-llc | 94% |
| Amalgam-pow | 93% |
| QTAG | 90% |
| Amalgam-parts | 78% |

The table shows that the only tagger that did not produce 90 per cent or more correct tags was the PARTS scheme of the AMALGAM Tagger. This means that it will not be considered in the future, at least not for the texts from *Invandrartidningen*.

From the table below we can see that it did not perform well for *Regeringsförklaringen* either, though, and therefore it will not be considered further, along with the LLC and POW schemes of the AMALGAM Tagger.

| *Regeringsförklaringen* (English version) | |
|---|---|
| Tagger/tagset: | percentage correct: |
| TreeTagger | 99% |
| Amalgam-upenn | 98% |
| Amalgam-lob | 97% |
| Amalgam-brown | 97% |
| Amalgam-sec | 96% |
| QTAG | 94% |
| Amalgam-ice | 94% |
| Amalgam-llc | 89% |
| Amalgam-pow | 88% |
| Amalgam-parts | 82% |

| *Scania* | |
|---|---|
| Tagger/tagset: | percentage correct: |
| TreeTagger | 95% |
| Amalgam-upenn | 94% |
| Amalgam-brown | 93% |
| Amalgam-llc | 92% |
| Amalgam-lob | 91% |
| Amalgam-ice | 91% |
| Amalgam-sec | 91% |
| Amalgam-pow | 91% |
| QTAG | 91% |
| Amalgam-parts | 77% |

Finally, the PARTS scheme of the AMALGAM Tagger did not produce a satisfying result on the *Scania* texts either (see table above) and from this we can conclude that it is not suitable for any of the mentioned text types.

The two taggers with the best results overall, i.e. for all text types put together, are the TreeTagger and the AMALGAM UPENN scheme with percentages above 96. It seems like a too coarse grained tagset like the one AMALGAM PARTS (20 tags) uses is too general to produce good results, but a slightly finer one, like the one used by TreeTagger and AMALGAM UPENN (45 tags) produces much better results. But seemingly, very fine grained tagsets like the AMALGAM BROWN with its 226 tags also perform well (95% correct overall). Prütz (1999b) has done some interesting work within the ETAP Project to simplify more complicated tagsets and comparing the results from training his Brill tagger with both the simplified tagset as well as its original.

It is very possible that the performance of QTAG would improve notably if the input text would be of another format, i.e. with one token per line instead of untokenized text. This because of the fact that the delimiters did not receive tags when putting in an untokenized text into QTAG. But when a test was performed later on to see if they would receive tags if separated from the words it was evident that this was the case. As it seems that delimiters are often tagged correctly, by simple calculation you can figure out that the percentage results for QTAG would improve.

Furthermore it has sometimes been hard to decide if the sentences have been tagged correctly. As Teubert (1996) points out, it is not an easy task to decide if the first element of for example 'language technology' (the equivalent to e.g. the Swedish 'Språkteknologi') is used as some kind of modifier (e.g. an adjective) or if it is the first constituent of a compound. It seems like the taggers have different 'opinions' about this, so when we have counted the percentage correct tags of the above tagsets we have

taken these kinds of differences into account.

When it comes to texts like *Invandrartidningen* and *Regeringsförklaringen* which contain a lot of specific Swedish phenomena, like Swedish institutions, Swedish proper names etcetera, one could suspect the translations of the Swedish texts to contain a lot of such occurrences. Thus the target texts will contain a lot of words that the tagger in question might not be able to interpret correctly. This might lead to a slightly lower percentage correct tags, because of misinterpretations and failures in recognising them as foreign words[5].

## 2.3 Tagset mapping

The next step after getting rid of the taggers performing below the 90 per cent level for each text type was to set up an equivalence table to map the tagsets to each other. This equivalence table will make it possible to compare the tagger results in a more specific manner than by only the percentage correct tags.

Mapping of tagsets might at a quick glance be considered an easy task, but it has proven to be far from trivial. If the tagsets are very similar, like the AMALGAM SEC and AMALGAM LOB tagsets which are practically the same, there is of course no problem, but the mapping of a complicated tagset like AMALGAM ICE from the AMALGAM SEC tagset is quite a complicated task. As Teufel (1995) points out, the biggest problem in mapping tagsets to each other does not occur when we have `1:1` (renaming) or `n:1` (the 'source' tagset makes finer distintions than the target annotation scheme) cases, but when (and they occur frequently) we come across `1:n` (the target tagset has a finer distinction that is not supported by the source tagset) and `n:m` (overlap between tag classes) cases.

In brief the mapping has been carried out by looking at the different tagsets, their tags, the description of the tags and examples of word forms tagged with the particular tags, and comparing the 'source' tagset with the 'target' tagsets. The tables of the different AMALGAM tagsets[6] have proven to be an invaluable help. Santorini (1990) contains valuable additional information on the Upenn tagset and for QTAG a simple list of the tags and a description of them (including some example word forms) has been used[7]. Frequently it has also been necessary to compare the tagged test texts to see which tags correspond to which.

[5]Only applicable to tagging schemes containing a specific tag for foreign words, though.

[6]<http://www.scs.leeds.ac.uk/amalgam/tagsets/tagmenu.html>

[7]<http://www-clg.bham.ac.uk/oliver/java/qtag/BLT-tagset.html>

Mapping problems of the types mentioned in the previous section has proven to be quite frequent indeed while trying to set up an equivalence table of the different English tagsets. As mentioned already some of the tagsets are very similar and do not cause a lot of problem, but there are also tagsets that are not very similar, e.g. the ICE, LLC and POW tagsets are examples of tagsets that use somewhat different annotation schemes. LLC and POW were designed for annotating transcribed dialogues, so that might be an answer to why they look and function the way they do.

Since the AMALGAM SEC tagging scheme produced the best result for the texts of *Invandrartidningen* we chose it as the source tagset from which the mapping to the other eight tagsets would be done. For the other text types a separate equivalence table will be set up where the mapping will be from TreeTagger instead. This because of the fact that it performed best for both *Regeringsförklaringen* and the *Scania* texts.

In setting up the equivalence table for *Invandrartidningen* a lot of examples of the `1:n` case mentioned above have been observed, i.e. the case where the target tagset has a finer distinction than the source tagset. One example is where AMALGAM SEC has one tag for foreign words (`&FW`) and the BROWN scheme has 50 tags (!). Another example is the more or less fine distinction of verbs. The SEC scheme has one tag (`BEM`) for present tense, 1st person singular of the verb "to be" (i.e. *am 'm*), whereas the ICE scheme has six different tags: `AUX(pass,pres)` (*am*), `AUX(pass,pres,encl)` (*'m*), `AUX(prog,pres)` (*am*), `AUX(prog,pres,encl)` (*'m*), `V(cop,pres)` (*am*) and `V(cop,pres,encl)` (*'m*). One might think that one way to deal with this problem is to start out from the scheme with the largest tagset, but this will however only solve some of the `1:n` cases since the smaller tagset might have a finer distinction in some of the categories. This is the case with common and proper nouns for ICE and SEC (4 to 30 tags). The ICE scheme has however separate tags for adjectives that function as nouns, e.g. *the oppressed* etcetera, and this distinction does not exist in the SEC scheme. Furthermore there are tags, like `NC` (cited word) in SEC that do not occur in any of the other so far mapped tagsets. Of course we have found a lot of examples of simply renaming the tags (`1:1`) and cases where the source tagset has a finer distinction than the target one (`n:1`) and these will mostly not cause any problems at all. The most problematic case is however the one where there is an overlap between tag classes (`n:m`). This is the case when e.g. a proper noun in one tagset is interpreted as a common noun in another. The example that Teufel (1996) brings up is the case when *anybody* is (wrongly) put into the common noun category in the Upenn tagset, but regarded as the pronoun it really is in the others. An additional problem is the way

the different tagsets interpret genitives and negated verbs as well as other 'combined' words like `I've` and `that'll`. Some of the schemes have special tags for (most of) these occurrencies (ICE, LLC, SEC, LOB, POW) whilst others tokenize these words into two words and give them two tags (BROWN, UPENN, TreeTagger, QTAG). It also has to be taken into consideration if it is correct to tag certain combined words of the type described above as two words. Teufel (1995) argues about this as well when she claims that the Upenn tagging of 'Peter's' in *Peter's house* (`Peter/NP 's/POS`) should be regarded as one nominal item whereas 'he's' in *he's not at home* (`he/PP 's/VBZ`) should be kept as two words.

Six of the tagging schemes have been mapped from the SEC tagging scheme and two remain to be mapped. Other remaining tasks in the evalutation of taggers will be discussed below.

Having set up the equivalence table, one can start comparing and thus evaluating the taggers in an automatic way by inserting the table into the program `compTags.pl` (see below), which compares texts tagged with the source and target tagsets and makes calculations on how similar they are and how much they differ. This will hopefully facilitate the selection of which tagger to use for which text types. A comparison of if one tagger is systematically better than another in those cases where they differ should also be carried out, i.e. does one tagger tag proper nouns (incorrectly) as common nouns whereas the other tagger tags them correctly? By comparing such differences one could eventually sum up the advantages and disadvantages of a particular tagger and simply choose the best one. Another approach is to decide if it is worth combining two or more taggers in some way or another and see if an even better result can be obtained. This is a very interesting approach and it is somewhat related to the work by Prütz (1999b) mentioned earlier.

All the above steps of testing, mapping, and evaluating will also be performed for the taggers of the other ETAP languages.

# 3 Comparing and combining German taggers

This section describes the implementation of a Perl program, written to simplify the evalution of two German POS taggers, Morphy and TreeTagger, and the use of this program for comparing and combining the two taggers.

The program compares the POS tags for each token in a text and summarizes the results. The purpose of the program was to determine which tagger to use for POS tagging the ETAP texts, but also for investigating

whether a combination of taggers would be able to outperform its individual members.

## 3.1 Collecting taggers

The first step was to collect available German taggers.

Three taggers were found: *Morphy* (Lezius et al. 1998), *TreeTagger* (Schmid 1994) and *QTag* (Mason 1997). These are briefly described below.

### 3.1.1 Morphy

Morphy is a tool for doing morphological analysis, part-of-speech tagging and context-sensitive lemmatization of German texts.

Morphy has a lexicon of 324.000 word forms and also has the ability to process compound nouns. All possible lemmata and morphological description is given for each word. Any ambiguities are resolved by the disambiguator or tagger. If a word cannot be recognized its part of speech is predicted by a guesser which uses statistical data from German suffix frequencies. Morphy consist of three major parts, the lexical system, the generation module and the analysis module. Morphy stores for each word its base form and its inflectional class. This gives 324.000 word forms. This lexicon can also be added more words by the user.

From the root form and inflectional class the generation produces all inflectional forms. The algorithms include vowel mutation, infixation and deletion.

The analysis module determines for each word form its root, part-of-speech and, in appropriate cases, gender, case number, person, tense and comparative degree. The module also segments compound nouns, by matching the longest rule. Ambigous words cannot be treated at this stage. This is done by the disambiguator or tagger. If a word cannot be recognized its part of speech is predicted by a guesser which uses statistical data.

Morphy uses one of two tagsets, a large set containing 1000 tags and a small set containing 46+6 tags. (Lezius 1995).

The tagger provides about 85% accuracy with the large tagset and 96% with the small tagset (Lezius et al. 1998).

### 3.1.2 TreeTagger

TreeTagger is a probabilistic part-of-speech tagger. Apart from other probabilistic taggers, TreeTagger uses a decision tree to estimate transition prob-

abilites.

The reason for using a decision tree is to avoid problems that other probabilistic taggers based on first- or second order Markov Models. The decision tree automatically determines the appropriate size of the context which is used to estimate the transition pobabilities.

TreeTagger achieves 96.36% accuracy on Penn-Treebank data, in comparison with a trigram tagger which achieves 96.06% accuracy on the same data. (Schmid 1994).

### 3.1.3 QTag

QTag is a language independent, probabilistic tagger, roughly based on the Hidden Markov Model. It is written in Java to achive portability.

QTag consists of two parts. A dictionary, where words and possible tags and frequencies are stated, and a matrix of tag sequences and frequencies. All you need to start is a (manually) tagged training corpus in order to generate the language specific resource files needed.

After considering the pros and cons of the taggers we decided only to use Morphy and TreeTagger. Unfortunately, the German resource file that was available for QTag was not very good, as it got trained on just one 19th century novel.

## 3.2 Implementation

### 3.2.1 Running the taggers

The first thing that was done was to run the two taggers to get an idea of how they worked and performed. The taggers were run on four different texts. Two from the Scania Corpus and two from The Swedish Statement of Government Policy Corpus. Overall it seemed in this stage that the two performed the same result, if not Morphy performed a little bit better. The results from this pre-evaluation can be found in the Appendix.

Further, while Morphy had its own statistical summary, TreeTagger was able to handle SGML tags, which for Morphy had to be removed (using a simple *sed*-command) before tagging.

### 3.2.2 Tokenization

While Morphy came with its own tokenizer, TreeTagger did not. The tagger required each token to be tagged to be on a separate line. Therefore a tokenizer had to be built.

In order to compare the two taggers they had to be parallel. This meant that each tagged token to be compared had to be the same for the both taggers. Therefore the two tokenizers had to perform the same result.

By default, lines were split on any word delimiter except punctuation mark. This was due to that Morphy removed some of the punctuation marks and some it let be. Overall it tagged each part of the abbreviation, e.g. it tagged Art.Nr. as:

```
Art  SUB  NOM  SIN  FEM
.    SZE
nr   ABK
```

And z.B. as:

```
z  ABK
b  ABK
```

This meant that lines could not be split on punctuation marks since some should be tagged as punctuation marks and some not, but as part of the abbreviation. Therefore, these cases had to be explictly handled by the tokenizer.

Also, Morphy completely ignored some characters like "*". These cases also had to be explictly handled by the tokenizer.

### 3.2.3 Comparing tagged tokens

Now that the taggers tagged the same type and amount of tokens, they could be compared.

Since the two taggers had different tagsets, and the use of tags should be compared, equivalent tags had to be determined.

Morphy contained a much larger tagset. This meant a lot of many to one relations, particulary for verbforms. For some, equivalent tags could not be established at all. Other relation came apparent after the analysis and could be added at a later time.

This was done manually by going through the tagset specifications. Approximately 60 hashes where used to declare which tags were equivalent. For example:

```
$pos{"SUB"} = "NN"
$pos{"EIG"} = "NE"
$pos{"VER 1 SIN"} = "VVFIN"
$pos{"VER 2 SIN"} = "VVFIN"
```

The keys are the Morphy tagset and the values are from TreeTagger set.

There were however some special cases that had to be handled. Due to the fact that Morphy had a much more detailed tagset, there was a lot of information in the tagstrings that were redundant to get the equivalent tag. But, some information was crucial to obtain in order to establish the correct equivalent tag. Therefore a simple check was performed to get the information needed. These cases turned out to be mostly pronouns.

Another problem was the auxiliary verbs like 'sein', 'haben' and 'werden'. Morphy could distinguish these more than just as auxiliary verbs depending on the context, which TreeTagger couldn't. These were handled by specific cases.

### 3.2.4 Algorithm

The script was written in Perl. It reads one Morphy tagged file and one TreeTagger tagged file. The script creates a file containing the results. The algorithm is as follows:

```
- Read Morphy tagged file
- Retrieve tag/tagstring from line
- Lookup its equivalent tag in hashtable
- Compare equivalent tag with tag in TreeTagger file
- Print result
```

### 3.2.5 Output

The script makes a file containing the results. First are all tagged tokens, including those agreed and not agreed on. They is printed in the following notation:

**token Morphylemma Morphytag =(/)= TreeTaggertag TreeTaggerlemma**

Example:

```
Regierung SUB NOM SIN FEM == NN  Regierung
dazu ADV PRO =/= PAV dazu
```

Then there are some statistics, like number of taged tokens, number of same used tags.

Finally, all case were the taggers disagreed are printed. It has the following notation:

**Number Morphytag TreeTaggertag**

It says how many of the specific Morphy tags became TreeTagger tags. Below is some sample output:

```
Verantwortung Verantwortung SUB NOM SIN FEM == NN Verantwortung
dafür dafür ZUS =/= PAV dafür
, , SZK ==  $,  ,
dass dass KON UNT =/= ADV <unknown>
die der ART DEF NOM SIN FEM == ART  d
nächsten nächst ADJ DEF DAT SIN FEM == ADJA nah

################## Results ########################

2604 tagged tokens
Number of times taggers agreed [2415 / 92.7 %]
Number of times taggers disagreed [189 / 7.3 %]

#########################################################

####### Disagreements  ########
###############################

12 ART DEF NOM SIN FEM became PRELS
11 VER 3 PLU became VVINF
9 KON UNT became ADV
8 ADV PRO became PAV
```

## 3.3 Analysis

The analysis of the taggers was carried out according to the following procedure. One or two short texts from the various subcorpora were tagged with each of the taggers. Then ten sentences were picked out and the number of correct and incorrect tags in them counted. After this, a correspondence table was constructed the for the tag comparison program described above, and the program was used on the output of the taggers to make pairwise comparisons of the taggers.

### 3.3.1 German taggers

Of the three German taggers evaluated, one, QTAG, turned out to have too low accuracy. This is probably due to it having been trained on nineteenth century fiction (Oliver Mason, p.c.), while the ETAP texts are contemporary.

In the following table, the other two German taggers are compared, with respect to their performance on two text types, technical manuals from the Scania subcorpus, and political prose from the German translation of the Swedish Statement of Government Policy (SGP) of 1988 and

1996. Accuracy percentages are calculated as: ERROR COUNT/NO. OF TAGGED TOKENS.

| *tagger/tagset* | *Scania* | *SGP* |
|---|---|---|
| TreeTagger | 96.3% | 96.2% |
| Morphy/crisp | 90.4% | 93.8% |
| Morphy/fuzzy | 94.7% | 95.4% |

The 'crisp' and 'fuzzy' tagsets used with Morphy refer to the way tagging errors were counted; with the 'crisp' way of counting, the whole morphosyntactic description had to be correct, i.e., if any part of it was incorrect—e.g., if the case was given as 'dative' instead of 'nominative' (a fairly common error)—the error count would be increased by 1. In the 'fuzzy' case, however, a correct part of speech[8] together with an error or errors in gender, case, and number for nominal parts of speech, and person/number for finite verbs, only counted as 0.25 errors.

## 3.4 Combining German taggers

Now all pieces are in place. We have two German taggers, as well as a means for automatically comparing their output on the same text material. Now we can proceed to test whether the following hypotheses hold:

1. that there would be differences between the two taggers in the errors they made, and

2. that these differences would show some systematicity, which could be utilized to improve tagging accuracy by combining the two taggers.

These hypotheses were tested in an experiment, and both turned out to be confirmed. There were differences between the taggers (see the following table), and some of the differences turned out to be systematic (with the proviso that the material used is fairly small).

[8]Here we used, roughly, the part of speech inventory of the other tagger, so that, e.g., finite and infinite verbs were counted as different parts of speech, even though they have the common major POS "VER" according to Morphy.

| *corpus* | *Morphy correct* | *TreeTagger correct* | *neither correct* | *total* |
|---|---|---|---|---|
| RF | 101 (35.5%) | 176 (62.0%) | 7 (2.5%) | 284 |
| Scania | 86 (36.1%) | 139 (58.4%) | 13 (5.5%) | 238 |
| total | 187 (35.8%) | 315 (60.4%) | 20 (3.8%) | 522 |

*Tagger differences: Which tagger was right how often?*

Finding the systematic differences between POS taggers implies making a decision as to which variables should be taken into account, i.e. should provide the input parameters for the if–then rules which should be the result of the next step. This amounts to a hypothesis about which factors influence tagging performance, and our initial hypothesis has been that the following parameters are relevant:

- the individual tags themselves;
- disjunctions of tags, denoting linguistically natural categories, e.g., both common nouns and proper nouns are nouns, both verbs and adjectives are verbal words in many languages, etc.;
- the text type, in our case the technical text of the Scania corpus vs. the administrative-political text type of the SGP;

Using the lists of differences between taggers and the hypothesis about which parameters were likely to influence tagger performance, it was possible to formulate rules for choosing the output of the inferior tagger (Morphy) over that of the better tagger (TreeTagger) under certain, systematically recurring, conditions definable in linguistic terms. This is where our work differs from other investigations of tagger combining, where machine learning methods are used;[9] see Borin 2000b for a discussion of this and for more details on the experiment with combining German POS taggers.

[9]Machine learning methods tend to need large amounts of training material, i.e. material which is correctly tagged for part-of-speech, i.e. the kind of material which you tend *not* to have when applying existing taggers to new material

# 4 Conclusion

In conclusion, we may say that:

1. it is possible to use off-the-shelf POS taggers on new text material, but it is worthwhile evaluating them on the material first;

2. POS taggers can be combined using linguistically motivated rules, yielding a combination which works better than any member of the combination.

# References

## Websites

AMALGAM SITE:
<http://www.scs.leeds.ac.uk/amalgam/amalgam/amalghome.htm>

Multext: <ftp://issco-ftp.unige.ch/pub/multext/>

Ordklasstaggningstjänst (GU Språk & Data AB):
<http://www.gusd.holding.gu.se/>, choose *Ordklasstaggning*

QTAG: <http://www-clg.bham.ac.uk/oliver/java/qtag/>

TreeTagger:
<http://www.ims.uni-stuttgart.de/Tools/DecisionTreeTagger.html>

## Literature

Atwell, Eric, George Demetriou, John Hughes, Amanda Schiffrin, Clive Souter and Sean Wilcock 2000. A comparative evaluation of modern English corpus grammatical annotation schemes. *ICAME Journal* 24:7–23.

Borin, Lars 2000a. (with contributions by others) ETAP project status report. In: Lars Borin (ed.), *Working Papers in Computational Linguistics & Language Engineering 23. Reports from the ETAP project*. Department of Linguistics, Uppsala University. 1–20. ETAP status report

December 2000. Research report etap-rr-06 2000. Department of Linguistics, Uppsala University.

Borin, Lars 2000b. Something borrowed, something blue: rule-based combination of POS taggers. In: *Second International Conference on Language Resources and Evaluation. Proceedings. Volume 1*. Athens: ELRA. 21–26.

Daelemans, W., Zavrel, J., Berck, P. & S. Gillis 1996. MBT: a memory-based part of speech tagger-generator. *Proceedings of the Fourth Workshop on Very Large Corpora*, Copenhagen, Denmark, 14–27.

Lager, Torbjörn 1999. The $\mu$-TBL system: logic programming tools for transformation-based learning. *Proceedings of the Third International Workshop on Computational Natural Language Learning (CoNLL'99)*. Bergen: ACL.

Lezius, Wolfgang 1995. Algorithmen zum Taggen deutscher Texte. Arbeitsbericht. Universität Paderborn, Fachbereich 2 - Psychologie.

Lezius, Wolfgang, Reinhard Rapp and Manfred Wettler 1998. A freely available morphological analyzer, disambiguator, and context sensitive lemmatizer for German. *COLING-ACL'98. 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics. Proceedings of the Conference, Vol. I–II.* Montreal: Université de Montréal.

Mason, O. 1997. QTAG–A portable probabilistic tagger. Corpus Research, University of Birmingham.

Prütz, Klas 1999a. Sammanställning av en träningskorpus på svenska för träning av ett automatiskt ordklasstaggningssystem. In: Anna Sågvall Hein (ed.), *Working Papers in Computational Linguistics & Language Engineering 19. Reports from the ETAP project*. Department of Linguistics, Uppsala University. 1–15.

Prütz, Klas 1999b. Part-of-speech tagging for Swedish. In: Lars Borin (ed.), *Working Papers in Computational Linguistics & Language Engineering 20. Reports from the ETAP project: Tagging and alignment*. Department of Linguistics, Uppsala University. 11–15.

Sánchez León, F. & A.F. Nieto Serrano 1995. CRATER – Corpus resources and terminology extraction deliverable report 22 – public domain POS tagger for Spanish. The tagger can be obtained via: <ftp://ftp.lllf.uam.es/pub/CRATER/esT-tagger-1-0.tar.gz>

Santorini, B. 1990. Part-of-speech tagging guidelines for the Penn Treebank Project. Technical report MS-CIS-90-47, Department of Computer and Information Science, University of Pennsylvania.

Schmid, Helmut 1994. Probabilistic part-of-speech tagging using decision trees. *Proceedings of the International Conference on New Methods in Language Processing*. Manchester.

Teubert, Wolfgang 1996. Language Resources for Language Technology. In: Tufis, Dan (ed.), *Limbaj si Tehnologie*. Editura Academiei Române, Bukarest. 19–28.

Teufel, Simone 1995. A support tool for tagset mapping. *Proceedings of SIGDAT 1995*. Workshop in cooperation with EACL 95, Dublin. The report can be obtained at:
<http://www.cogsci.ed.ac.uk/∼simone/eacl95.ps.gz>

## Working Papers in Computational Linguistics & Language Engineering

Uppsala University, Department of Linguistics, Box 527, SE-751 20 Uppsala, Sweden.
URL: <http://www.ling.uu.se/> (e-mail: <info@ling.uu.se>)

No. 11 Reports from the SCARRIE Project, Editor: Anna Sågvall Hein.
*CORRIE-based Grammar Checking.*
Wedbjer Rambell, Olga: *Swedish Phrase Constituent Rules. A Formalism for the Expression of Local Error Rules for Swedish, Deliverable 6.3.3, 6.4 and 6.4.3*. 28 pp.
Wedbjer Rambell, Olga: *A Minor Grammar Checking Test for Swedish Using the Fragment Analysis Approach in CORRIE*. 26 pp.
Uppsala, December 1999.

No. 12 Reports from the SCARRIE Project, Editor: Anna Sågvall Hein.
*Chart-Based Grammar Checking in SCARRIE.*
Sågvall Hein, Anna, Starbäck, Per: *A Test Version of the Grammar Checker for Swedish, Deliverable 6.5.1*. 44 pp.
Sågvall Hein, Anna: *A Specification of the Required Grammar Checking Machinery, Deliverable 6.5.2*. 39 pp.
Sågvall Hein, Anna: *A Grammar Checking Module for Swedish, Deliverable 6.6.3*. 24 pp.
Starbäck, Per: *ScarCheck – a Software for Word and Grammar Checking*. 6 pp.
Weijnitz, Per: *Uppsala Chart Parser Light System Documentation*. 20 pp.
Uppsala, December 1999.

No. 13 Reports from the SCARRIE Project, Editor: Anna Sågvall Hein.
*Evaluating the Swedish SCARRIE Prototype.*
Sågvall Hein, Anna, Leif-Jöran Olsson, Bengt Dahlqvist, Erik Mats: *Evaluation Report for the Swedish Prototype, Deliverable 8.1.3*. 16 pp.
Ahlbom, Viktoria, Sågvall Hein, Anna: *Test Suites Covering the Functional Specifications of the Sub-components of the Swedish Prototype, Deliverable 7.1.3*. 28 pp.
Uppsala, December 1999.

No. 14 Reports from the PLUG Project, Editor: Anna Sågvall Hein.
Tiedemann, Jörg: *Parallel Corpora in Linköping, Uppsala and Göteborg (PLUG): The Corpus*. 13 pp.
Uppsala, December 1999.

No. 15 Reports from the PLUG Project, Editor: Anna Sågvall Hein.
Ahrenberg, Lars, Merkel, Magnus, Sågvall Hein, Anna, Tiedemann, Jörg: *Evaluation of LWA and UWA*. 28 pp.
Uppsala, December 1999.

No. 16 Reports from the PLUG Project, Editor: Anna Sågvall Hein.
Sågvall Hein, Anna: *The PLUG-project. Parallel Corpora in Linköping, Uppsala, Göteborg: Aims and Achievements*. 17 pp.
Uppsala, December 1999.

No. 17 Reports from the PLUG Project, Editor: Anna Sågvall Hein.
Tiedemann, Jörg: *Uplug – A Modular Corpus Tool for Parallel Corpora*. 16 pp.
Uppsala, December 1999.

No. 18 Reports from the ETAP Project, Editor: Anna Sågvall Hein.
*Converting, Aligning and Tagging for ETAP.*
Tjong Kim Sang, Erik: *Converting the SCANIA Framemaker Documents to TEI SGML*. 14 pp.
Tjong Kim Sang, Erik: *Aligning the Scania Corpus*. 7 pp.
Qiao, Hong Liang: *Comparing the Tagging Performance Between the AGTS and Brill Taggers*. 9 pp.
Uppsala, December 1999.

No. 19 Reports from the ETAP Project, Editor: Anna Sågvall Hein.
Prütz, Klas: *Sammanställning av en träningskorpus på svenska för träning av ett automatiskt ordklasstaggningssystem*.15 pp.
Uppsala, December 1999.

No. 20 Reports from the ETAP Project, Editor: Lars Borin.
*Tagging and Alignment.*
Borin, Lars: *Alignment and Tagging*. 10 pp.
Prütz, Klas: *Part-of-Speech Tagging for Swedish*. 5 pp.
Uppsala, December 1999.

No. 21 Reports from the ETAP Project, Editor: Lars Borin.
*Seeing Double: Using Parallel Corpora for Linguistic Research.*
Olsson, Leif-Jöran, Borin, Lars: *ETAP-WebTEq: a Web-Based Tool for Exploring Translation Equivalents on Word and Sentence Level in Multilingual Parallel Corpora*. 8 pp.
Borin, Lars, Prütz, Klas: *Through a Glass Darkly: Part of Speech Distribution in Original and Translated Text*. 22 pp.
Uppsala, December 2000.

No. 22 Reports from the ETAP Project, Editor: Lars Borin.
*Segmenting and Tagging Parallel Corpora.*
Oxhammar, Henrik, Borin, Lars: *Sentence Splitting and SGML Tagging*. 10 pp.
Bengtsson, Camilla, Borin, Lars, Oxhammar, Henrik: *Comparing and Combining Part of Speech Taggers for Multilingual Parallel Corpora*. 20 pp.
Uppsala, December 2000.

No. 23 Reports from the ETAP Project, Editor: Lars Borin.
Borin, Lars, with contributions by others:
*ETAP Project Status Report December 2000*. 20 pp.
Uppsala, December 2000.