Using Syntactic Constraints in Natural Language Disambiguation

Jorge Graña Gil

January 1999

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Elimination of ambiguities is a crucial task during the process of tagging a text in natural language. If we take in isolation, for instance, the word `time`, we can see that it has several possible tags in English: substantive, adjective, verb, ... However, if we examine the context in which the word appears, only one of the tags is possible. In addition, we are also interested in being able to give a tag to all the words that appear in a text, but are not present in our dictionary, and to guarantee that this tag is the correct one. A good performance at this stage will improve the viability of syntactic and semantic analysis.

One of the most common techniques for solving this kind of problem is the use of Hidden Markov Models [6]. This technique applies a stochastic procedure, which calculates Markov chains from a training corpus, and then uses these chains to tag new texts. Another technique is transformation-based error driven tagging. This method automatically produces a lexicon and a set of lexical and contextual transformation rules. If a given transformation yields a correct pair (word, tag), then the rule is applied in order to try to obtain the best tagging. This is the technique used for instance by Brill's tagger [5].

However, a small percentage of wrongly tagged words (2-3%) is a feature that is always present in pure stochastic systems. For this reason we support the idea of using these in combination with syntactic information, that is, with parsing techniques, and this is the aim of the present work.

To perform our experiments we need a parser and a reference corpus. In regard to the parser, we will use the SLPToolKit [1], an on-line bottom-up parser for stochastic context-free grammars that, in addition to the usual functionalities of standard SCFG parsers, is able to find the N-most probable parsings of the input sequence, to deal with multiple interpretations of sentences containing compound words, and to deal with out of vocabulary words. Furthermore, it has proved to be at least as efficient as the corresponding Earley-like or CYK-like parsers. In regard to the reference corpus, we will use the Susanne corpus [2], a treebank that comprises an approximately 150,000-word subset corpus of English.

The general idea is to divide the reference corpus into two parts, to extract a grammar from the first one, to parse the second one, to consider all the subtrees that we obtain for the non-parseable sentences, to design strategies (similar to the ones used in robust parsing) to combine those syntactic constraints, to apply them to retag those non-parseable sentences, to evaluate the performance of these new disambiguation techniques, and to compare these with the traditional ones mentioned above. The following chapters explain in detail all these steps.

Chapter 2

Reference corpus

The SUSANNE corpus [2] is the *treebank* that has been used as the reference corpus in our study. The SUSANNE corpus has been created, with the sponsorship of the *Economic and Social Research Council (UK)*, as part of the process of developing a comprehensive NLP-oriented taxonomy and annotation scheme for the (logical and surface) grammar of English: the SUSANNE scheme.

The SUSANNE scheme [3] attempts to provide a method of representing all aspects of English grammar which are sufficiently definite to be susceptible of formal annotation. It offers a scheme of categories and ways of applying them that make it practical for NLP researchers to register everything that occurs in real-life usage systematically and unambiguously, and for researchers at different sites to exchange empirical grammatical data without misunderstandings over local uses of analytic terminology.

The SUSANNE corpus has been produced largely manually, not as the output of an automatic parsing system. It itself comprises an approximately 150,000-word subset of the BROWN corpus of American English, annotated in accordance with the SUSANNE scheme. The finished SUSANNE parsing scheme has thus been developed on the basis of samples of both British and American English. It does not cover other languages, although it is hoped that its general principles may prove helpful in developing comparable taxonomies for these. It is oriented chiefly towards written language.

2.1 Structure of the SUSANNE corpus

The SUSANNE corpus consists of 64 files, each containing an annotated version of one 2,000-word text from the BROWN corpus. Files average is about 83 kilobytes in size, thus the entire corpus is about 5.3 megabytes. The file names are those of the respective BROWN texts, e.g. `A01`, `N18`. Sixteen texts are drawn from each of the following BROWN genre categories:

- `A` (press reportage).
- `G` (belles lettres, biography, memoirs).
- `J` (learned writing, mainly scientific and technical).
- `N` (adventure and Western fiction).

Each file has a line (terminating in a newline character) for each "word" of the original text. But "words" for SUSANNE purposes are often smaller than words in the ordinary orthographic sense, for instance punctuation marks and the apostrophe-s suffix are treated as separate words and assigned lines of their own.

Where text characters cannot be adequately represented directly within the character set, they are represented by entity names within angle brackets. Where possible these are drawn from *Standard*

Generalized Markup Language (SGML). For instance, `<eacute>` stands for lower-case `e` with acute accent. Symbols in angle brackets are used also to represent such things as typographical shifts, which for purposes of grammatical analysis are conveniently represented as items within the word-sequence: e.g. `<bital>` stands for *begin italics*.

Each line of a SUSANNE file has six fields separated by tabs. Each field on every line contains at least one character. The six fields on each line are:

1. **Reference field**. The reference field contains nine bytes which give each line a reference number that is unique across the SUSANNE corpus, e.g. `N06:1530t`. The first three bytes (here `N06`) are the file name; the fourth byte is always a colon; bytes 5 to 8 (here `1530`) are the number of the line in the *Bergen I* version of the BROWN corpus on which the relevant word appears (BROWN line numbers normally increment in tens, with occasional odd numbers interpolated); and the ninth byte is a lower-case letter differentiating successive words that appear on the same BROWN line. SUSANNE lines are lettered continuously from `a`, omitting `l` and `o`.)

2. **Status field**. The status field contains one byte. The letters `A` and `S` show that the word is an *abbreviation* or *symbol*, respectively, as defined by BROWN corpus codes.

   The SUSANNE corpus aims to reflect the incidence of errors found in real-life written English, and therefore to reproduce those errors which stem from the original texts on which the Brown Corpus was based, while correcting errors that were introduced into the BROWN corpus during the process of corpus-construction. When an error (or apparent error) reflects the form found in the original publication, it is preserved in the SUSANNE corpus flagged by an `E` in the status field. Otherwise, the SUSANNE corpus restores the text of the original publication and the status field ignores the error. Where errors are original, wordtagging and grammatical analysis is applied to the erroneous text as best it can be by analogy with correct forms.

   On the great majority of lines, to which none of these three categories apply, the status field contains a hyphen character.

3. **Wordtag field**. The SUSANNE wordtag set is based on the LANCASTER tagset [4]. Additional grammatical distinctions have been drawn in this set, and these are indicated by suffixing lower-case letters to the Lancaster tags. For instance, `revealing` is tagged `VVG` (present participle of verb) in the LANCASTER scheme, but as `VVGt` (present participle of transitive verb) in the SUSANNE scheme. Apart from the lower-case extensions, the wordtags are normally identical to the LANCASTER tags: punctuation marks are assigned alphabetical tags beginning with `Y` (e.g. `YC` for comma), and the dollar sign which appears in some LANCASTER tags for genitive words is replaced by `G` (e.g. `GG` for the apostrophe-s suffix), so that the modified LANCASTER tags always consist wholly of alphanumeric characters, beginning with two capital letters.

   The tag `YG` appears in the wordtag field to represent a "ghost" or a *trace*, that is, the logical position of a constituent which has been shifted elsewhere, or deleted, in the surface grammatical structure.

   The SUSANNE tagset comprises 425 distinct wordtags, and is shown in appendix A.

4. **Word field**. The word field contains a segment of the text, often coinciding with a word in the orthographic sense but sometimes, as noted above, including only part of an orthographic word. In general the word field represents all and only those typographical distinctions in the original documents which were recorded in the BROWN corpus, though in certain cases the SUSANNE corpus has gone behind the BROWN corpus to reconstruct typographical details omitted from BROWN.

   Certain characters have special meanings in the word field, as follows:

- `+` (occurs only as first byte of the word field) shows that the contents of the field were not separated in the original text from the immediately preceding text segment by whitespace (e.g. in the case of a punctuation mark, or part of a hyphenated sequence split over successive SUSANNE lines).
- `-` shows that the line corresponds to no text material (it represents the *trace* for a grammatically moved element).
- `<...>` (enclose entity names for special typographical features, as discussed above).

5. **Lemma field**. The lemma field shows the dictionary headword of which the text word is a form: the field shows base forms for words which are inflected in the text, and eliminates typographical variations (such as sentence initial capitalization) which are not inherent to the word but relate to its use in context. In the case of words to which the dictionary form concept is inappropriate, e.g. numerals and punctuation marks, the lemma field contains a hyphen.

6. **Parse field**. The contents of the sixth field represent the central information of the SUSANNE corpus. They code the grammatical structure of texts as a sequence of labelled trees, having a leaf node for each corpus line.

   Each text is treated as a sequence of *paragraphs* separated by *headings*. A *paragraph* normally coincides with an ordinary orthographic paragraph; a *heading* may consist of actual verbal material, or may be merely a typographical paragraph division, symbolized `<minbrk>` in the word field. Conceptually, the structure of each paragraph or heading is a labelled tree with root node labelled `O` (`Oh` for a heading), and with a leaf node labelled with a wordtag for each SUSANNE word or trace, i.e. each line of the corpus. There will commonly be many intermediate labelled nodes.

   Such a tree is represented as a bracketed string in the ordinary way, with the labels of non-terminal nodes written inside both opening and closing brackets (that is, to the right of opening brackets and to the left of closing brackets). This bracketed string is then adapted as follows for inclusion in successive SUSANNE parse fields. Wherever an opening bracket immediate follows a closing bracket, the string is segmented, yielding one segment per leaf node; and within each such segment, the sequence `[ wordtag ]`, representing the leaf node, is replaced by a full stop. Thus each parse field contains exactly one full stop, corresponding to a terminal node labelled with the contents of the wordtag field, sometimes preceded by labelled opening bracket(s) and sometimes followed by labelled closing bracket(s), corresponding to higher tagmas which begin or end with the word on the line in question.

   In total the parse-trees of SUSANNE comprise 267,046 nodes, of which 4,383 are roots and 156,584 are leaves. Notation for non-terminal node labels in the SUSANNE scheme is shown in appendix B.

In order to show the general aspect of all these data, we include a small set of lines taken from the SUSANNE corpus:

```
...
A01:0010b    -   AT     The            the            [O[S[Nns:s.
A01:0010c    -   NP1s   Fulton         Fulton         [Nns.
A01:0010d    -   NNL1cb County         county         .Nns]
A01:0010e    -   JJ     Grand          grand          .
A01:0010f    -   NN1c   Jury           jury           .Nns:s]
A01:0010g    -   VVDv   said           say            [Vd.Vd]
A01:0010h    -   NPD1   Friday         Friday         [Nns:t.Nns:t]
A01:0010i    -   AT1    an             an             [Fn:o[Ns:s.
A01:0010j    -   NN1n   investigation  investigation  .
A01:0020a    -   IO     of             of             [Po.
A01:0020b    -   NP1t   Atlanta        Atlanta        [Ns[G[Nns.Nns]
A01:0020c    -   GG     +&lt;apos&gt;s  -              .G]
```

```
A01:0020d    -   JJ     recent         recent        .
A01:0020e    -   JJ     primary        primary       .
A01:0020f    -   NN1n   election       election      .Ns]Po]Ns:s]
A01:0020g    -   VVDv   produced       produce       [Vd.Vd]
A01:0020h    -   YIL    <ldquo>        -             .
A01:0020i    -   ATn    +no            no            [Ns:o.
A01:0020j    -   NN1u   evidence       evidence      .
A01:0020k    -   YIR    +<rdquo>       -             .
A01:0020m    -   CST    that           that          [Fn.
A01:0030a    -   DDy    any            any           [Np:s.
A01:0030b    -   NN2    irregularities irregularity  .Np:s]
A01:0030c    -   VVDv   took           take          [Vd.Vd]
A01:0030d    -   NNL1c  place          place         [Ns:o.Ns:o]Fn]Ns:o]Fn:o]S]
A01:0030e    -   YF     +.             -             .O]
...
```

2.2 Features of the SUSANNE corpus

This section describes exactly the features of the SUSANNE corpus. All of the PERL scripts referenced here are shown in appendix E. Assuming `susanne` to be a symbolic link to a directory containing the SUSANNE corpus, we can obtain a corpus and its features from the files in the SUSANNE treebank, by the following commands:

```
cat susanne/A* susanne/G* susanne/J* susanne/N*
    | susanne_treebank_to_tagged_corpus.prl > SUSANNE_full_TAGGED_CORPUS
cat SUSANNE_full_TAGGED_CORPUS | corpus_to_lexicon.prl | sort
    | tagged_words_to_lexicon.prl > SUSANNE_full_LEXICON
cat SUSANNE_full_LEXICON | features_lexicon.prl
features_corpus.prl SUSANNE_full_LEXICON SUSANNE_full_TAGGED_CORPUS
```

The features of `SUSANNE_full_TAGGED_CORPUS` are the following:

- The corpus has 153,514 words. The size of the file is 1,470,342 characters. The syntax of each line is:

  ```
  form_1/tag_1 form_2/tag_2 ... form_n/tag_n ./YF
  ```

  Lines could also end with any other punctuation mark, and thus each line is a sentence. The corpus has 4,383 sentences, that is 35 words per sentence, on average.

- The features of `SUSANNE_full_LEXICON`, the lexicon formed by all the forms that appear in the corpus, are the following:

  - 14,959 forms with 1 tag,
  - 1,311 forms with 2 tags,
  - 125 forms with 3 tags,
  - 36 forms with 4 tags,
  - 13 forms with 5 tags,
  - 9 forms with 6 tags,
  - 1 form with 7 tags,
  - 3 forms with 8 tags,
  - 1 form with 9 tags and
  - 1 form with 14 tags.

That is, 16,459 different forms, with 18,273 possible tags, and corresponding to 10,416 different lemmas. If we calculate the *percentage of ambiguous forms* and the *average number of tags per form*, we obtain:

$$\%\, ambiguous\ forms = \frac{\#\, ambiguous\ forms}{\#\, forms} \times 100 = \frac{16459 - 14959}{16459} \times 100 = 9.11\ \%$$

$$\#\, average\ of\ tags\ per\ form = \frac{\#\, tags}{\#\, forms} = \frac{18273}{16459} = 1.11\ tags\ per\ form$$

- It is much more interesting to calculate the same features directly with all the words in `SUSANNE_full_TAGGED_CORPUS`, and we obtain the following numbers:
  - 79,428 words with 1 tag,
  - 31,059 words with 2 tags,
  - 5,583 words with 3 tags,
  - 7,573 words with 4 tags,
  - 12,177 words with 5 tags,
  - 3,397 words with 6 tags,
  - 3,119 words with 7 tags,
  - 7,460 words with 8 tags,
  - 2,880 words with 9 tags and
  - 838 words with 14 tags.

  That is, 153,514 different words, with 389,019 possible tags.

$$\%\, ambiguous\ words = \frac{\#\, ambiguous\ words}{\#\, words} \times 100 = \frac{153514 - 79428}{153514} \times 100 = 48.26\ \%$$

$$\#\, average\ of\ tags\ per\ word = \frac{\#\, tags}{\#\, words} = \frac{389019}{153514} = 2.53\ tags\ per\ word$$

Here we have the general aspect of the files `SUSANNE_full_TAGGED_CORPUS`:

```
...
The/AT Fulton/NP1s County/NNL1cb Grand/JJ Jury/NN1c said/VVDv Friday/NPD1 an/AT1
investigation/NN1n of/IO Atlanta/NP1t <apos>s/GG recent/JJ primary/JJ election/NN1n
produced/VVDv <ldquo>/YIL no/ATn evidence/NN1u <rdquo>/YIR that/CST any/DDy
irregularities/NN2 took/VVDv place/NNL1c ./YF
...
```

and `SUSANNE_full_LEXICON`:

```
...
Aah UH
Aaron NP1m
Abatuno NP1s
Abbas NP1s
Abe NP1m
Abelson NP1s
Abner NP1m
```

```
About RGi II
Above II RL
Abraham NP1m
Abreaction NN1n
Academy NNJ1c
Accademia FW
Accidental JJ
According II21
Accordingly RR
Accounts NN2
Acres NN2
Act NN1c
Acting VVGv JJ
...
```

2.3 Extracting resources from the SUSANNE corpus

This section describes exactly what kind of linguistisc resources we have extracted from the SUSANNE corpus. All of the PERL scripts referenced here are also shown in appendix E.

- We obtain `SUSANNE_full_TAG_TABLE.slptoolkit`, the tag table of SUSANNE corpus by the command

  ```
  cat susanne/A* susanne/G* susanne/J* susanne/N* | cut -f3 | sort -u
      | gen_slptoolkit_tag_table.prl > SUSANNE_full_TAG_TABLE.slptoolkit
  ```

  Here we have the general aspect of the file `SUSANNE_full_TAG_TABLE`:

  ```
  1:APPGf
  2:APPGh1
  3:APPGh2
  4:APPGi1
  5:APPGi2
  6:APPGm
  7:APPGy
  8:AT
  9:AT1
  10:AT1e
  11:ATn
  12:BTO21
  13:BTO22
  14:CC
  15:CC31
  ...
  ```

- We obtain `SUSANNE_full_LEXICON.slptoolkit`, the lexicon derived from all the forms that appear in the corpus, in SLPTOOLKIT syntax, by the following commands:

  ```
  cat susanne/A* susanne/G* susanne/J* susanne/N*
      | susanne_treebank_to_tagged_corpus.prl | corpus_to_lexicon.prl | sort
      | lexicon_to_rules.prl | sort | rules_to_stochastic_grammar.prl > temp
  susanne_grammar_to_slptoolkit_lexicon.prl SUSANNE_full_TAG_TABLE.slptoolkit temp
      | sort > SUSANNE_full_LEXICON.slptoolkit
  rm temp
  ```

  Here we have the general aspect of the file `SUSANNE_full_LEXICON.slptoolkit`, where the first column is the form, the third one is the number of the tag in the tag-table, and the last one is the probability of the rule $tag \rightarrow form$:

```
...
a.m.||296||0.2
a<hyphen>tall||334||0.00367647058823529
aback||325||0.0333333333333333
abandoned||133||0.00133511348464619
abandoned||391||0.0099009900990099
abandoned||399||0.00552486187845304
ability||167||0.00144092219020173
ablaze||125||0.2
able||133||0.00133511348464619
aboard||325||0.0333333333333333
aborigines||172||0.0012970168612192
about||106||0.0151515151515152
about||318||0.5
about||333||1
above||106||0.0151515151515152
above||325||0.0333333333333333
abruptly||334||0.00367647058823529
absently||334||0.00367647058823529
...
```

We have decided not to use the lemma, and therefore columns two (lemma) and four (frequency) are empty in our case.

- It is also possible to obtain the same lexicon in binary format, by the following commands and steps:

```
cut -d'|' -f1 SUSANNE_full_LEXICON.slptoolkit > mylex.tree
cut -d'|' -f3 SUSANNE_full_LEXICON.slptoolkit > mylex.morph
cut -d'|' -f5 SUSANNE_full_LEXICON.slptoolkit > mylex.proba
```

Now we create a file called for instance `mylex`, containing these lines:

```
#
mylex.tree
no
mylex.morph
no
mylex.proba
425
0
```

By the command

```
creelex mylex SUSANNE_full_LEXICON-bin
```

we obtain several files. Among these, we have the file `SUSANNE_full_LEXICON-bin.slplex` enumerating all the different parts of the binary lexicon. In our case, we manually edit it adding the path `/users/grana/TreeBank/data/` to the name of the other files (in order to be able to access to the binary lexicon from any directory), and replacing lines 4 and 6 with `nolemma` and `nofreq`, to obtain:

```
SLPLex lexical file v1.0 11/97 (for binary format lexicon)
/users/grana/TreeBank/data/SUSANNE_full_LEXICON-bin.tree
/users/grana/TreeBank/data/SUSANNE_full_LEXICON-bin.graph
nolemma
/users/grana/TreeBank/data/SUSANNE_full_LEXICON-bin.morph
```

```
nofreq
/users/grana/TreeBank/data/SUSANNE_full_LEXICON-bin.proba
425
0
```

In order to check whether the binary lexicon generates the same forms as before, we can do this:

```
listlexique SUSANNE_full_LEXICON-bin.slplex > temp
cmp temp SUSANNE_full_LEXICON.slptoolkit
```

Finally, we remove all the intermediate steps and create a symbolic link to the main file:

```
rm temp mylex mylex.tree mylex.morph mylex.proba
ln -s SUSANNE_full_LEXICON-bin.slplex lexicon
```

- The first real transformation that we are going to perform with the SUSANNE treebank is to integrate in only one line all the syntactic and lexical information available for each sentence, keeping in the parsing the notation already introduced, that is, `[` and `]` each time a subtree starts and ends, respectively.

  However, as we can see in appendix B, the SUSANNE treebank contains non-terminal symbols for structures of higher level than a sentence (paragraphs and titles, for instance). Therefore, the next step is to take each parsing and remove all that is necessary (paragraph and title marks, for instance) in order to keep only the subparsings that correspond to real sentences, that is, subparsings with a root non-terminal starting with `S`, `F`, `T`, `W`, `A`, `Z` or `L`.

  Our main objective at this moment is to extract a grammar for our experiments from the SUSANNE corpus. As we have seen, the SUSANNE treebank contains two kind of sentences: traced and non-traced. Traced sentences can be useful, for instance, to study another kind of linguistic phenomenon in which we make reference in a non-explicit way to other components of the sentence, or even of other sentences, such as anaphora or cataphora. However, in our case, if we use this kind of sentences during the extraction process of the grammar, we could obtain non-terminal symbols not directly linked to real linguistic components, or we could even obtain a non-context-free grammar, and our parser would not be able to deal with it. Therefore, we prefer to split the SUSANNE corpus in two parts: one with non-traced sentences, from which we will extract our grammar, and another one with the traced sentences, which we will use as a bank for experiments.

  All these steps can be done by the commands:

```
cat susanne/A* susanne/G* susanne/J* susanne/N*
    | susanne_treebank_to_susanne_parsing.prl > temp
extract_real_sentences.prl SUSANNE_full_TAG_TABLE.slptoolkit temp
    | sort -u > SUSANNE_full_PARSING
rm temp
divide_susanne_parsing_into_non_traced_and_traced.prl
    SUSANNE_full_PARSING SUSANNE_notrace_PARSING SUSANNE_trace_PARSING
```

  Here we have the general aspect of the files `SUSANNE_notrace_PARSING`:

```
...
[ Fa [ CSf For CSf ] [ Ds:s [ DD1q each DD1q ] [ Po [ IO of IO ] [ Np [ DD2i these
DD2i ] [ NN2 lines NN2 ] Np ] Po ] Ds:s ] [ Vz [ VVZv meets VVZv ] Vz ] [ YTL
<bital> YTL ] [ N:o [ FOx Q FOx ] N:o ] [ YTR <eital> YTR ] [ P:p [ II in II ] [ Np
[ MC three MC ] [ NN2 points NN2 ] [ YC , YC ] [ REX namely REX ] [ N@ [ MC two MC ]
[ NN2 points NN2 ] [ P [ II on II ] [ FOx g FOx ] P ] [ Ns+ [ CC and CC ] [ MC1 one
MC1 ] [ NNL1n point NNL1n ] [ P [ II on II ] [ Ms [ MC1 one MC1 ] [ Po [ IO of IO ]
```

```
[ Np [ AT the AT ] [ JJ multiple JJ ] [ NN2 secants NN2 ] Np ] Po ] Ms ] P ] Ns+ ]
N@ ] Np ] P:p ] Fa ]
...
```

and SUSANNE_trace_PARSING:

```
...
[ Fa [ CSf For CSf ] [ Ni:s [ PPH1 it PPH1 ] Ni:s ] [ Vz [ VVZt includes VVZt ] Vz ]
[ Np:o173 [ AT the AT ] [ JJ emotional JJ ] [ NN2 ties NN2 ] [ Fr [ CST that CST ]
[ s173 [ YG - YG ] s173 ] [ V [ VV0v bind VV0v ] V ] [ Np:o [ NN2 men NN2 ] Np:o ]
[ P:q [ IIt to IIt ] [ Ns [ APPGh2 their APPGh2 ] [ NN1c homeland NN1c ] Ns ] P:q ]
Fr ] [ Np+:175 [ CC and CC ] [ AT the AT ] [ JJ complex JJ ] [ NN2 motivations NN2 ]
[ Fr [ CST that CST ] [ s175 [ YG - YG ] s175 ] [ V [ VV0v hold VV0v ] V ] [ Ns:o
[ AT1 a AT1 ] [ JJ large JJ ] [ NNJ1c group NNJ1c ] [ Po [ IO of IO ] [ NNc people
NNc ] Po ] Ns:o ] [ R:w [ RL together RL ] R:w ] [ P:j [ IIa as IIa ] [ Ns [ AT1 a
AT1 ] [ NN1c unit NN1c ] Ns ] P:j ] Fr ] Np+:175 ] Np:o173 ] Fa ]
...
```

- After this, we translate the file containing the parsing for the non-traced sentences to the SLPToolKit syntax, by the command:

```
susanne_parsing_notrace_to_slptoolkit_parsing.prl SUSANNE_full_TAG_TABLE.slptoolkit
    SUSANNE_notrace_PARSING > SUSANNE_notrace_PARSING.slptoolkit
```

and we obtain the file SUSANNE_notrace_PARSING.slptoolkit, with the following aspect:

```
...
[ X [ Fa [ :33 For ] [ Ds_s [ :59 each ] [ Po [ :121 of ] [ Np [ :74 these ] [ :172
lines ] ] ] ] [ Vz [ :403 meets ] ] [ :422 <bital> ] [ N_o [ :96 Q ] ] [ :423
<eital> ] [ P_p [ :106 in ] [ Np [ :146 three ] [ :172 points ] [ :408 , ] [ :302
namely ] [ N@ [ :146 two ] [ :172 points ] [ P [ :106 on ] [ :96 g ] ] [ Ns+ [ :14
and ] [ :147 one ] [ :182 point ] [ P [ :106 on ] [ Ms [ :147 one ] [ Po [ :121 of ]
[ Np [ :8 the ] [ :133 multiple ] [ :172 secants ] ] ] ] ] ] ] ] ] ]
...
```

There are several reasons for generating this new file. First, we need to add a new non-terminal symbol, X, which will be the starting symbol of our grammar. It is also necessary to change some characters which are special to the parser in the name of the non-terminals, and replace the word tags by their corresponding number in the tag-set table. Finally, we also need to make these parsings available in the same output format as that produced by the tool, in order to obtain a reference that is easily comparable with later trees.

- To obtain a stochastic grammar in SLPToolKit format from the notrace part of the SUSANNE corpus, we do the following:

```
cat SUSANNE_notrace_PARSING | susanne_parsing_notrace_to_rules.prl | sort |
    rules_to_stochastic_grammar.prl > SUSANNE_notrace_GRAMMAR
susanne_grammar_to_slptoolkit_grammar.prl SUSANNE_full_TAG_TABLE.slptoolkit
    SUSANNE_notrace_GRAMMAR > SUSANNE_notrace_GRAMMAR.slptoolkit
```

Before using this grammar, we will check whether it has cycles or not, because a cycle could make the parser enter a never ending loop. This checking can be performed by a very simple operation with matrixes. Given that we are working with a bottom-up parser, cycles can be produced only by unitary rules, that is, by rules with only one non-terminal symbol in both the right hand side and the left hand side of the arrow. There exist 206 unitary rules, involving 187 non-terminal symbols. Then we build a square matrix of $187 \times 187 = 34,969$ cells, in such a way

that given any two non-terminal symbols X and Y, the cell $U(X, Y)$ contains the probability of the rule $X \rightarrow Y$. Therefore, 206 of these cells contain the probabilities of the unitary rules we are considering, while the rest of the cells are zeros. If we know that

$$\sum_{i=0}^{\infty} x^i = 1 + x + x^2 + \ldots = \frac{1}{1-x}$$

then we calculate

$$V = \sum_{i=0}^{\infty} U^i = (I - U)^{-1}$$

where I is the identity matrix of order 187. If we now calculate $U \times V$, or, in other words, $V - I$, we obtain the matrix whose cells contain, for any two non-terminal symbols X e Y, the probability of $X \stackrel{*}{\Rightarrow} Y$, that is, the probability of X generating Y in a finite number of steps. Therefore, if values different from zero exist in the diagonal of $U \times V$, we have found cycles.

After applying this procedure, we detected only one cycle caused by the rules $Ot \rightarrow Nns$ and $Nns \rightarrow Ot$. The first of these rules appears many times throughout the sentences of the corpus, while the second appears only once at this concrete point of the corpus:

```
...
G22:0450j - AT The the [Nns[Ot[Np.
G22:0460a - JJ New new [Nns.
G22:0460b - NP1t York York .Nns]
G22:0460c - NNT2 Times time .Np]Ot]Nns]Po]
...
```

Since we have assumed that this is due to a transcription error in the corpus, we have manually replaced the subtree `[ Nns [ Ot [ ... ] Ot ] Nns ]` by `[ Ot [ Nns [ ... ] Nns ] Ot ]`. After rerunning the extraction of the rules, this cycle disappears and we obtain a grammar that is usable by our parser. This grammar contains 17,669 rules, 1,525 non-terminal symbols, and 28,117 nodes. In order to show the general aspect of the rules, we include a small set of lines directly taken from the grammar:

```
X -> Fa (0.002796)
X -> Fa% (0.000233)
X -> Fa_121 (0.000233)
X -> Fc (0.000466)
X -> Ff (0.000233)
X -> Fr (0.000233)
X -> L (0.009320)
X -> L! (0.000233)
X -> L+ (0.000233)
X -> L? (0.000233)
X -> L?+ (0.000466)
X -> S (0.916356)
X -> S! (0.001165)
X -> S!+ (0.000233)
X -> S% (0.000233)
X -> S* (0.011650)
X -> S*+ (0.000699)
X -> S+ (0.029590)
X -> S? (0.017008)
X -> S?+ (0.000932)
X -> S@ (0.000699)
```

```
X -> S_129 (0.000233)
X -> S_133 (0.000233)
X -> S_145 (0.000233)
X -> S_149 (0.000233)
X -> S_151 (0.000233)
X -> S_205 (0.000233)
X -> S_209 (0.000233)
X -> S_223 (0.000233)
X -> Tb (0.000466)
X -> Tb! (0.000233)
X -> Tb? (0.000466)
X -> Tg (0.002563)
X -> Tg+ (0.000233)
X -> Ti (0.000466)
X -> Tn (0.000466)
A -> :27 Ni_s Vzb P_r (0.105263)
A -> :27 Ns_S Vzp (0.0526316)
A -> :27 P_p (0.0526316)
A -> :27 R_t (0.0526316)
A -> :27 R_t Jh_e (0.0526316)
A -> :27 Ti_z (0.0526316)
A -> :27 Vg Fn_o (0.0526316)
A -> :27 Vn P_p (0.105263)
A -> :27 Vn P_q (0.0526316)
A -> :27 Vn P_r (0.0526316)
A -> :27 Vn P_u (0.0526316)
A -> :27 Vn Pb_a (0.315789)
...
```

- It is also possible to obtain the same grammar in binary format, by the following commands and steps:

  ```
  compilgram -P 426 SUSANNE_notrace_GRAMMAR.slptoolkit SUSANNE_notrace_GRAMMAR-bin
  ```

  We obtain several files. Among them, we have `SUSANNE_notrace_GRAMMAR-bin-conv.slplex`, `SUSANNE_notrace_GRAMMAR-bin-gram.slplex` and `SUSANNE_notrace_GRAMMAR-bin.slpgram` enumerating all the different parts of the binary grammar. In our case, we manually edit them adding the path `/users/grana/TreeBank/data/` to the name of the other files (in order to be able to access to the binary grammar from any directory), and replacing lines 4 and 6 with `nolemma` and `nofreq`, to obtain, for `SUSANNE_notrace_GRAMMAR-bin-conv.slplex`:

  ```
  SLPLex lexical file v1.0 11/97 (for binary format lexicon)
  /users/grana/TreeBank/data/SUSANNE_notrace_GRAMMAR-bin-conv.tree
  /users/grana/TreeBank/data/SUSANNE_notrace_GRAMMAR-bin-conv.graph
  nolemma
  /users/grana/TreeBank/data/SUSANNE_notrace_GRAMMAR-bin-conv.morph
  nofreq
  /users/grana/TreeBank/data/SUSANNE_notrace_GRAMMAR-bin-conv.proba
  1950
  0
  ```

  for `SUSANNE_notrace_GRAMMAR-bin-gram.slplex`:

  ```
  SLPLex lexical file v1.0 11/97 (for binary format lexicon)
  /users/grana/TreeBank/data/SUSANNE_notrace_GRAMMAR-bin-gram.tree
  /users/grana/TreeBank/data/SUSANNE_notrace_GRAMMAR-bin-gram.graph
  nolemma
  ```

```
/users/grana/TreeBank/data/SUSANNE_notrace_GRAMMAR-bin-gram.morph
nofreq
/users/grana/TreeBank/data/SUSANNE_notrace_GRAMMAR-bin-gram.proba
1950
0
```

and for `SUSANNE_notrace_GRAMMAR-bin.slpgram`:

```
SLPLex binary grammar file v1.0 07/98 (for binary format lexicon)
426
/users/grana/TreeBank/data/SUSANNE_notrace_GRAMMAR-bin-conv.slplex
/users/grana/TreeBank/data/SUSANNE_notrace_GRAMMAR-bin-gram.slplex
```

Finally, we create a symbolic link to the main files:

```
ln -s SUSANNE_notrace_GRAMMAR.slptoolkit grammar
ln -s SUSANNE_notrace_GRAMMAR-bin.slpgram grammar-bin
```

- To obtain tagged and untagged corpora from the notrace and from the trace parts of the SUSANNE corpus, we do the following:

```
cat SUSANNE_notrace_PARSING | susanne_parsing_to_tagged_corpus.prl
    > SUSANNE_notrace_TAGGED_CORPUS
cat SUSANNE_notrace_TAGGED_CORPUS | tagged_corpus_to_untagged_corpus.prl
    > SUSANNE_notrace_UNTAGGED_CORPUS
cat SUSANNE_trace_PARSING | susanne_parsing_to_tagged_corpus.prl
    > SUSANNE_trace_TAGGED_CORPUS
cat SUSANNE_trace_TAGGED_CORPUS | tagged_corpus_to_untagged_corpus.prl
    > SUSANNE_trace_UNTAGGED_CORPUS
```

The features of `SUSANNE_notrace_TAGGED_CORPUS` are the following: the corpus has 4,292 sentences and 77,275 words, that is 18 words per sentence, on the average. The size of the file is 751,359 characters.

The features of the lexicon formed by all the forms that appear in the corpus, are the following:

- 10,938 forms with 1 tag,
- 864 forms with 2 tags,
- 87 forms with 3 tags,
- 27 forms with 4 tags,
- 9 forms with 5 tags,
- 5 forms with 6 tags,
- 2 form with 7 tags,
- 1 forms with 8 tags and
- 2 form with 9 tags.

That is, 11,935 different forms, with 13,150 possible tags. If we calculate the *percentage of ambiguous forms* and the *average number of tags per form*, we obtain:

$$\%\, ambiguous\ forms = \frac{\#\, ambiguous\ forms}{\#\, forms} \times 100 = \frac{11935 - 10938}{11935} \times 100 = 8.35\ \%$$

$$\#\, average\ of\ tags\ per\ form = \frac{\#\, tags}{\#\, forms} = \frac{13150}{11935} = 1.10\ tags\ per\ form$$

It is much more interesting to calculate the same features directly with all the words in `SUSANNE_notrace_TAGGED_CORPUS`, and we obtain the following numbers:

- 42,550 words with 1 tag,
- 13,004 words with 2 tags,
- 9,225 words with 3 tags,
- 2,175 words with 4 tags,
- 2,174 words with 5 tags,
- 2,015 words with 6 tags,
- 1,539 words with 7 tags,
- 2,574 words with 8 tags and
- 2,019 words with 9 tags.

That is, 77,275 different words, with 177,429 possible tags.

$$\%\, ambiguous\ words = \frac{\#\, ambiguous\ words}{\#\, words} \times 100 = \frac{77275 - 42550}{77275} \times 100 = 44.93\ \%$$

$$\#\, average\ of\ tags\ per\ word = \frac{\#\, tags}{\#\, words} = \frac{177429}{77275} = 2.30\ tags\ per\ word$$

The features of `SUSANNE_trace_TAGGED_CORPUS` are the following: the corpus has 2,188 sentences and 60,759 words, that is 28 words per sentence, on the average. The size of the file is 593,735 characters.

The features of the lexicon formed by all the forms that appear in the corpus, are the following:

- 9,322 forms with 1 tag,
- 687 forms with 2 tags,
- 70 forms with 3 tags,
- 21 forms with 4 tags,
- 8 forms with 5 tags,
- 3 forms with 6 tags,
- 2 form with 7 tags,
- 1 forms with 8 tags and
- 1 form with 14 tags.

That is, 10,115 different forms, with 11,084 possible tags. If we calculate the *percentage of ambiguous forms* and the *average number of tags per form*, we obtain:

$$\%\, ambiguous\ forms = \frac{\#\, ambiguous\ forms}{\#\, forms} \times 100 = \frac{10115 - 9322}{10115} \times 100 = 7.84\ \%$$

$$\#\, average\ of\ tags\ per\ form = \frac{\#\, tags}{\#\, forms} = \frac{11084}{10115} = 1.06\ tags\ per\ form$$

It is much more interesting to calculate the same features directly with all the words in `SUSANNE_trace_TAGGED_CORPUS`, and we obtain the following numbers:

- 34,329 words with 1 tag,
- 8,957 words with 2 tags,

- 3,580 words with 3 tags,
- 2,232 words with 4 tags,
- 4,984 words with 5 tags,
- 2,268 words with 6 tags,
- 2,820 words with 7 tags,
- 1,236 words with 8 tags and
- 353 words with 14 tags.

That is, 60,759 different words, with 145,009 possible tags.

$$\%\, ambiguous\ words = \frac{\#\, ambiguous\ words}{\#\, words} \times 100 = \frac{60759 - 34329}{60759} \times 100 = 43.50\,\%$$

$$\#\, average\ of\ tags\ per\ word = \frac{\#\, tags}{\#\, words} = \frac{145009}{60759} = 2.39\ tags\ per\ word$$

Figure 2.1 graphically summarizes the process we have followed to extract all the resources from the SUSANNE corpus, together with their basic features.

The next chapter explains carefully how we have used the parser and all the resources obtained here, not only to test their quality, but also to evaluate the performance of the newly designed disambiguation techniques.

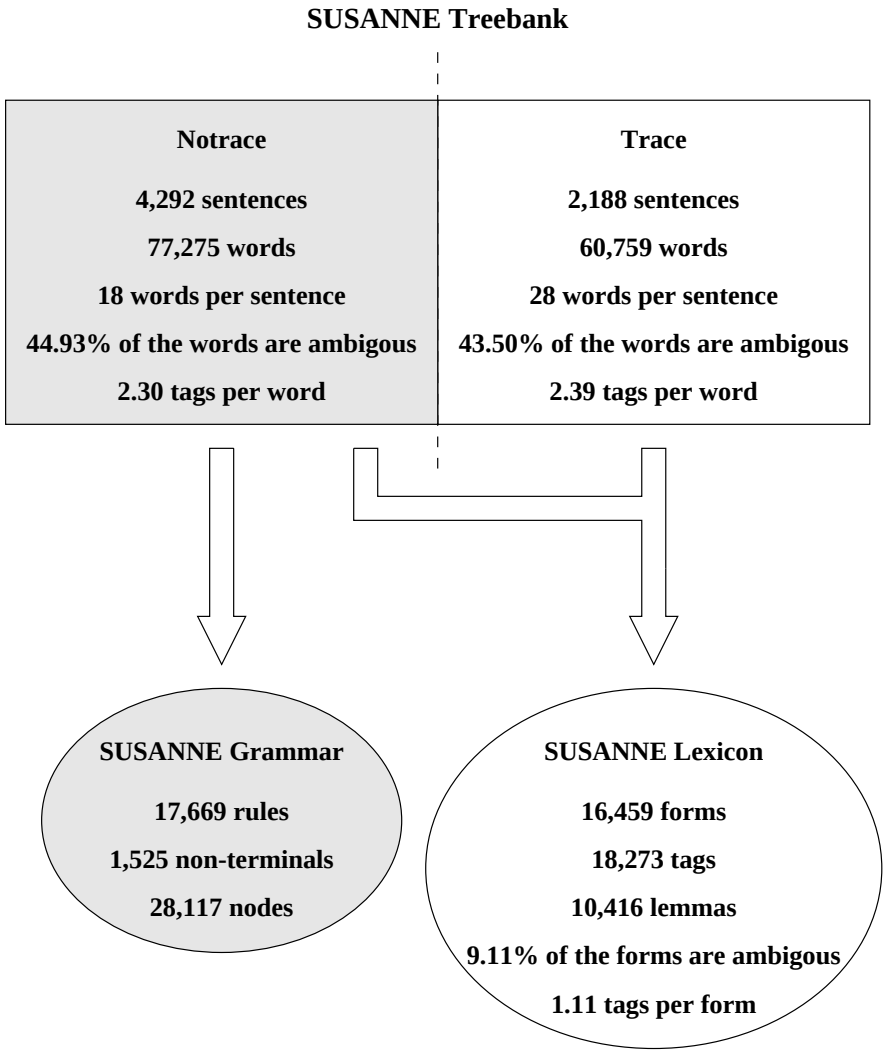


Figure 2.1: Splitting SUSANNE corpus and extracting SUSANNE grammar and SUSANNE lexicon

Chapter 3

Using the SLPToolKit with the Susanne corpus

To use all the generated resources with the SLPToolKit, in order to analyze real sentences, we can use one of these two commands:

```
anagram -K -P -i p -s lexicon grammar
anagram -K -P -i p -s -cg lexicon grammar-bin
```

The `-K` option deactivates automatic spelling correction, not needed because we are working with a lexicon generated from the whole corpus, and thus there is no unknown word. The `-P` option makes the parser generate in the output only those trees whose root is the starting symbol of the grammar. The `-i p` option means iterative output with parenthesis, against the shared trees provided by default. The `-s` option means silence mode, that is, no information messages. The `-cg` option allows us to use the compiled version of the grammar, in order to load it into the memory faster.

3.1 Parsing `Notrace` corpus with Susanne grammar

The first step is to analyze the coverage of the Susanne grammar on the same corpus from which it has been extracted We do this by the following command (as in the previous chapters, all of the Perl scripts referenced here are shown in appendix E):

```
test_slptoolkit_on_notrace.prl SUSANNE_notrace_PARSING.slptoolkit
    SUSANNE_notrace_UNTAGGED_CORPUS results/results_on_notrace
```

Of course, all the sentences in the `Notrace` corpus are parseable by the Susanne grammar, because all the rules involved in the trees of the reference corpus are present in the grammar. However, many other different interpretations can arise, that is, the reference tree could not be the only one for each sentence. Therefore, it is very interesting to consider the following two questions:

- what is the number of ambiguities generated by the grammar for each sentence in the `Notrace` corpus,
- and which is the rank of the reference tree inside the set of trees provided by the parser for each sentence.

The file `results/results_on_notrace` contains all the information that we need to study both items. With regard to the number of ambiguities, the concrete data after processing appear in section C.1, and summarized in figure 3.1. The most relevant feature of this distribution of values is that the median is between 7 and 8 interpretations per sentence.

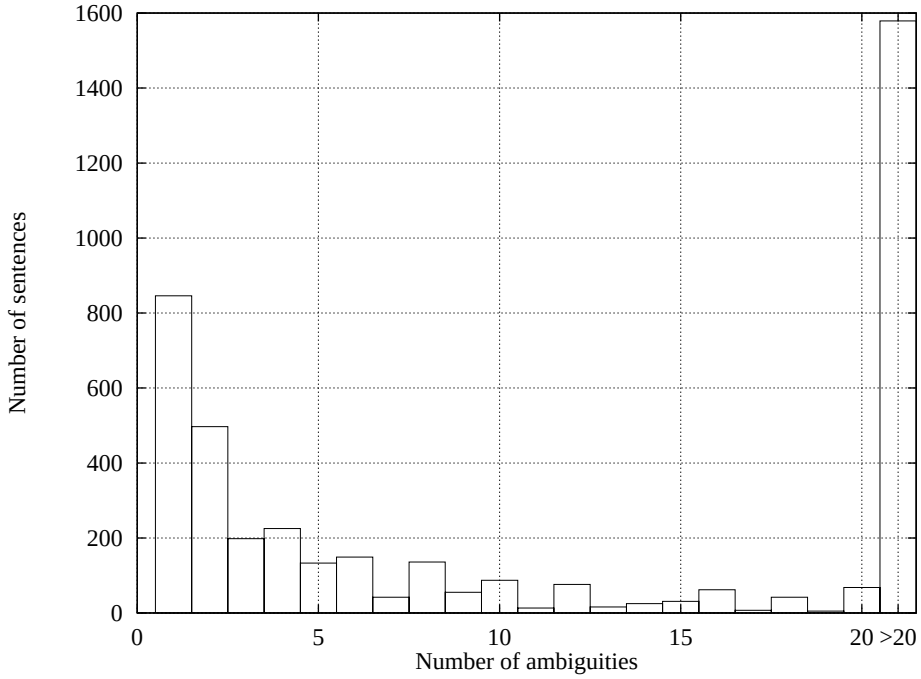


Figure 3.1: Ambiguities with SUSANNE grammar on `Notrace` corpus

With regard to the success of the ranking of parsings, the concrete data after processing appear in section C.2, and summarized in figure 3.2. The most relevant feature is that the parsing with the highest probability is equal to the reference tree for 77.703% of the sentences, which is a higher value than the usual success rates. This fact points up several things: the quality of the tagging in the reference corpus is very good, the extraction process of the grammar has been correct, the stochastic parsing model can be applied to this corpus, and our tagger implements this paradigm correctly.

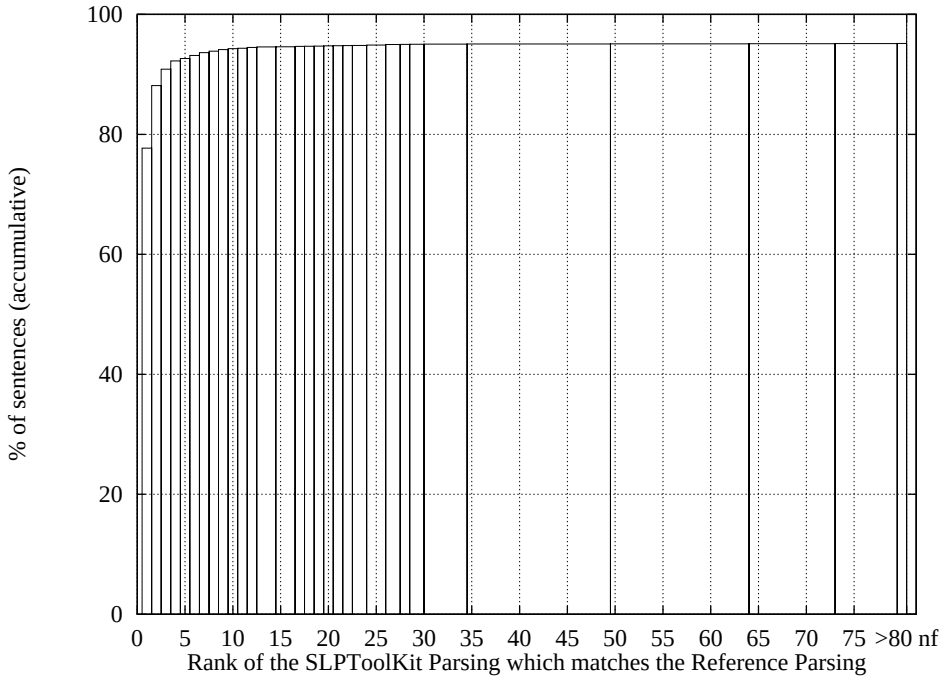


Figure 3.2: Success with SUSANNE grammar on `Notrace` corpus

3.2 Tagging Notrace corpus with SUSANNE grammar

However, our immediate interest is focussed only on the disambiguation process, and we want to know whether, in general, the syntactic information can help in this process or not. We have a syntactic resource available in the form of a grammar and before continuing with our experiment, we are going to perform the following test: for all the sentences in the Notrace corpus, we consider only the most probable tree among all the trees provided by the parser, even if is different from the reference tree, and we only look at the tags that this most probable tree gives to each word, and we compare those tags with the tags in the reference. We do this by the following set of commands:

```
tagged_corpus_to_sequences_of_tags.prl SUSANNE_full_TAG_TABLE.slptoolkit
    SUSANNE_notrace_TAGGED_CORPUS > SUSANNE_notrace_TAGGED_CORPUS.tags
test_slptoolkit_on_notrace_tags.prl SUSANNE_notrace_TAGGED_CORPUS.tags
    SUSANNE_notrace_UNTAGGED_CORPUS results/results_on_notrace_tags
```

But first, we will define exactly what we are going to measure. When we compare the retagged text and the reference corpus, we calculate the following counters:

- OOVF+ (Out Of Vocabulary Form Successful)
- OOVF- (Out Of Vocabulary Form Failure)
- NAF+ (Non-Ambiguous Form Successful)
- NAF- (Non-Ambiguous Form Failure)
- AF+ (Ambiguous Form Successful)
- AF- (Ambiguous Form Failure)

From them, we also calculate:

- #OOVF (Number of Out Of Vocabulary Forms) = (OOVF+) + (OOVF-)
- #NAF (Number of Non-Ambiguous Forms) = (NAF+) + (NAF-)
- #AF (Number of Ambiguous Forms) = (AF+) + (AF-)
- #F (Number of Forms) = (#OOVF) + (#NAF) + (#AF)

And then, we will use two rates, S1 (precision) and S2 (recall), in order to obtain a definite idea of the performance in the tagging process. S1 (precision) measures the global level of success. Therefore, in general, it is calculated as:

$$S1\ (precision) = \frac{OK}{OK + ERROR} \times 100$$

and, in our case, as:

$$S1\ (precision) = \frac{(OOVF+) + (NAF+) + (AF+)}{\#F} \times 100$$

S2 (recall) is a rate that belongs to the terminology used in *information extraction*, and measures the level of success when the system has a real chance of success. In general, it is calculated as:

$$S2\ (recall) = \frac{OK + ERROR}{OK + ERROR + SILENCE} \times 100$$

and, in our case, as:

$$S2\,(recall) = \frac{(OOVF+) + (AF+)}{\#F - \#NAF} \times 100$$

A good tagging system should present a high value (close to 100, the highest value), not only in the rate of global success, but in both rates.

However, in order to get an idea about whether the results will be good or not, we must compare them with any other tagger. In this case, we have used BRILL system, which we have trained on the `Notrace` corpus, and we have retagged the same text by using the `SUSANNE_full_LEXICON`. Therefore, in principle, these should be the best use conditions for BRILL system: to retag the same corpus used in the training, and with a lexicon in which there are no out-of-vocabulary forms.

Table 3.1 shows the performance in the tagging process for both systems. We can observe that the values for the SUSANNE grammar working as a tagger are very close to a 100% success rate. This allows us to support the idea that syntactic information plays an important role in the disambiguation process and could help a lot. Therefore, we can now continue our experiment.

Tagging `Notrace` corpus		
	BRILL system	SUSANNE grammar
OOVF+	0	0
OOVF-	0	0
NAF+	40945	40945
NAF-	0	0
AF+	32853	36034
AF-	3477	296
S1	95.50	99.61
S2	90.43	98.18

Table 3.1: BRILL system vs. SUSANNE grammar tagging `Notrace` corpus

3.3 Parsing `Trace` corpus with SUSANNE grammar

But before we do this, let us see what happens when we try to parse the sentences in the `Trace` corpus with SUSANNE grammar. We can do this by the command:

```
test_slptoolkit_on_trace.prl SUSANNE_trace_UNTAGGED_CORPUS results/results_on_trace
```

With regard to the number of ambiguities and to the successful of the ranking of parsings, the concrete data after processing appear in sections C.3 and C.4, respectively. The most relevant feature is that only 60 sentences in the `Trace` are parsed by the SUSANNE grammar. Therefore, the coverage of the SUSANNE grammar on the sentences with traces is very bad, but against this we now have a good bank of 2,128 sentences available, on which we can experiment with robust parsing and with disambiguation driven by the syntax.

The next step is to remove the parsed sentences from the `Trace` corpus by the commands:

```
tagged_corpus_to_sequences_of_tags.prl SUSANNE_full_TAG_TABLE.slptoolkit
    SUSANNE_trace_TAGGED_CORPUS > SUSANNE_trace_TAGGED_CORPUS.tags
cut -f2 results/results_on_trace | egrep -n ^[^0] | cut -d":" -f1 > temp
remove_lines_by_number.prl temp SUSANNE_trace_PARSING
    > SUSANNE_trace_np_PARSING
remove_lines_by_number.prl temp SUSANNE_trace_TAGGED_CORPUS
```

```
    > SUSANNE_trace_np_TAGGED_CORPUS
remove_lines_by_number.prl temp SUSANNE_trace_TAGGED_CORPUS.tags
    > SUSANNE_trace_np_TAGGED_CORPUS.tags
remove_lines_by_number.prl temp SUSANNE_trace_UNTAGGED_CORPUS
    > SUSANNE_trace_np_UNTAGGED_CORPUS
rm temp
```

and calculate the forest of possible partial subtrees for each of the 2,128 sentences non-parseable with traces. However, we are interested only in the disambiguation process, and, for each subtree, we want to consider only its sequence of word tags, and not the whole subtree. This set of sequences of tags for all those sentences is not a great amount of information, but its calculation process implies navigating through the shared subtrees stored in the cells of the parsing table, and this does mean a great amount of time. For this reason, we prefer to perform a new partition, this time inside the `Trace` corpus, considering the number of words for each sentence:

- The first portion, that we will call `Trace_np_25` corpus, is formed by 1,045 sentences, and these are the non-parseable sentences with traces and with 25 words or less. The time taken to compute the forests of sequences of tags for these sentences was about 25 minutes in the host `liasun13.epfl.ch (SunOS liasun13 5.5.1 Generic_103640-08 sun4u sparc SUNW,Ultra-2)`.
- The second portion, that we will call `Trace_np_50` corpus, is formed by 951 sentences, and these are the non-parseable sentences with traces and with a word count of between 26 and 50. The time taken to compute the forests of sequences of tags for these sentences was about 63 hours in the same host.
- The third portion is formed by the remaining 132 sentences, and these are the non-parseable sentences with traces and with more than 50 words. Due to the great amount of time needed to compute the forests of sequences of tags for the sentences in the second part, this third group has been removed from the experiment.

This partition and the forests of sequences of tags are performed by the following commands[1]:

```
divide_in_two_by_number_of_words.prl 25 SUSANNE_trace_np_UNTAGGED_CORPUS
    SUSANNE_trace_np_25_UNTAGGED_CORPUS temp
divide_in_two_by_number_of_words.prl 50 temp SUSANNE_trace_np_50_UNTAGGED_CORPUS
    temp2
rm temp temp2

divide_in_two_by_number_of_words.prl 25 SUSANNE_trace_np_TAGGED_CORPUS.tags
    SUSANNE_trace_np_25_TAGGED_CORPUS.tags temp
divide_in_two_by_number_of_words.prl 50 temp SUSANNE_trace_np_50_TAGGED_CORPUS.tags
    temp2
rm temp temp2

forest -K -i f -s -cg lexicon grammar-bin < SUSANNE_trace_np_25_UNTAGGED_CORPUS
   > temp
combine_forest_reference.prl temp SUSANNE_trace_np_25_TAGGED_CORPUS.tags
   > SUSANNE_trace_np_25_FOREST
rm temp

forest -K -i f -s -cg lexicon grammar-bin < SUSANNE_trace_np_50_UNTAGGED_CORPUS
   > temp
combine_forest_reference.prl temp SUSANNE_trace_np_50_TAGGED_CORPUS.tags
   > SUSANNE_trace_np_50_FOREST
rm temp
```

[1] `forest` is not a PERL script, but a C program that directly uses the resources implemented in the SLPTOOLKIT library of functions, and a piece of its code is also shown in appendix E

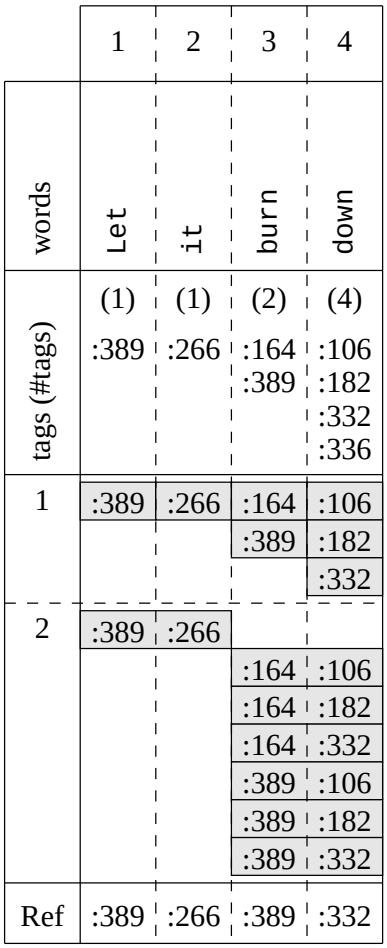
The size of the file `SUSANNE_trace_np_25_FOREST` is 3,590,277 bytes, and contains 94,057 different subsequences of tags for the 1045 sentences in `Trace_np_25` corpus. The size of the file `SUSANNE_trace_np_50_FOREST` is 8,831,545 bytes, and contains 201,684 different subsequences of tags for the 951 sentences in `Trace_np_50` corpus.

But in order to have a better idea about the aspect of these forests of sequences of tags, we have taken 8 sentences from the `Trace_np_25` corpus:

```
trace_np_sample_1: Let it burn down

trace_np_sample_2: That <apos>s all I ask

trace_np_sample_3: Each is still glorified as a national hero

trace_np_sample_4: All his people ask for is no more war

trace_np_sample_5: That <apos>s what they <apos>ll expect you to do

trace_np_sample_6: There is a tangible feeling in the air of revulsion toward
                   politics

trace_np_sample_7: The mystique of sex , combined with marijuana and jazz , is
                   intended to provide a design for living

trace_np_sample_8: <ldquo> I <apos>d just turned on the ignition when there was a
                   big flash and I was lying on the driveway <rdquo> , he said
```

The forests and the traced trees that appear in the reference corpus are shown in figures 3.3 to 3.18. It is also important to say that this is not the only information provided by the generating process of the forest. Besides this, for each one of the sequences of tags represented, we know the number of subtrees that cover the same portion of sentence with the same tags, and we also know the probability of the most probable of them, as we can see in appendix D.

All this information forms a very good working document that will allow us in the next chapter to think about the design of strategies for combining these syntactic constraints imposed by these sequences of tags, in order to try to solve the disambiguation problem.



	1	2	3	4
words	Let	it	burn	down
tags (#tags)	(1) :389	(1) :266	(2) :164 :389	(4) :106 :182 :332 :336
1	:389	:266	:164	:106
			:389	:182
				:332
2	:389	:266		
			:164	:106
			:164	:182
			:164	:332
			:389	:106
			:389	:182
			:389	:332
Ref	:389	:266	:389	:332

Figure 3.3: Forest for `trace_np_sample_1` (a 4-word sentence with a cross product of 8 possible tags)

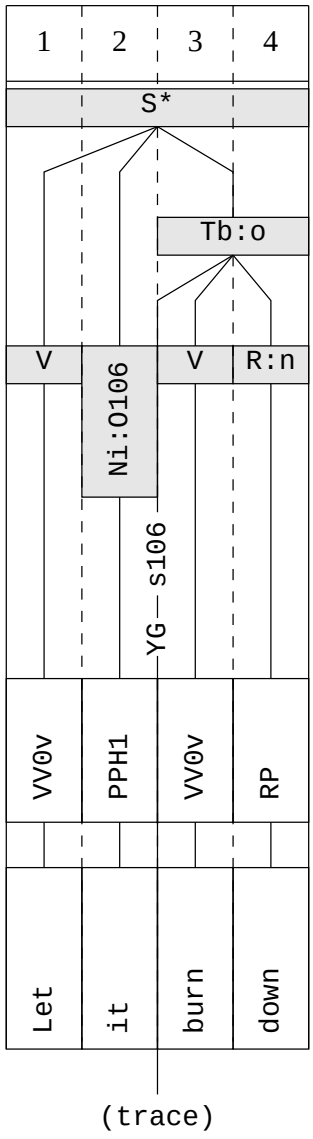
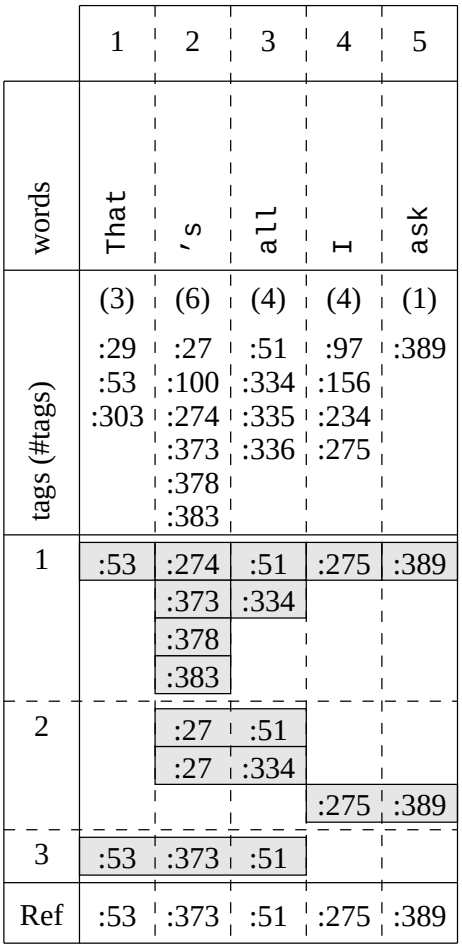


Figure 3.4: Reference tree for `trace_np_sample_1` (the 4-word sentence in figure 3.3)



	1	2	3	4	5
words	That	's	all	I	ask
tags (#tags)	(3) :29 :53 :303	(6) :27 :100 :274 :373 :378 :383	(4) :51 :334 :335 :336	(4) :97 :156 :234 :275	(1) :389
1	:53	:274 :373 :378 :383	:51 :334	:275	:389
2		:27 :27	:51 :334	:275	:389
3	:53	:373	:51		
Ref	:53	:373	:51	:275	:389

Figure 3.5: Forest for `trace_np_sample_2` (a 5-word sentence with a cross product of 288 possible tags)

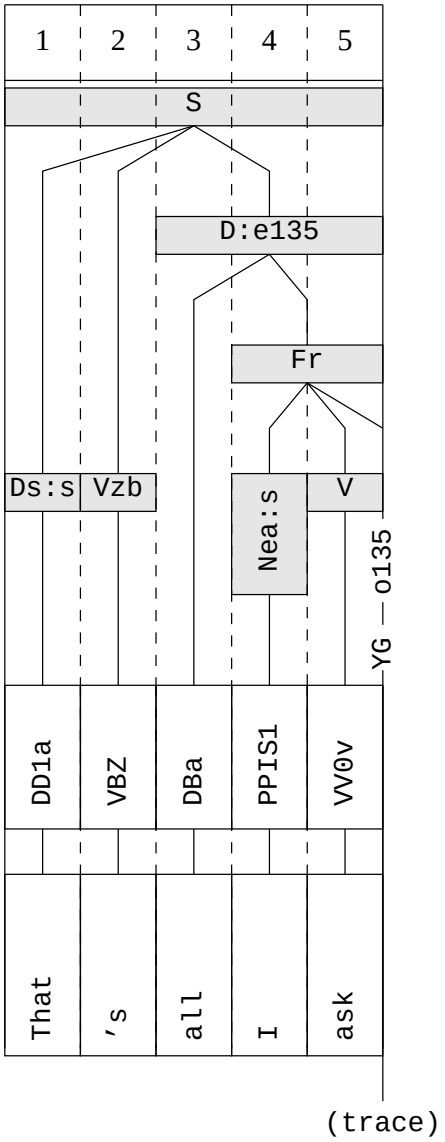
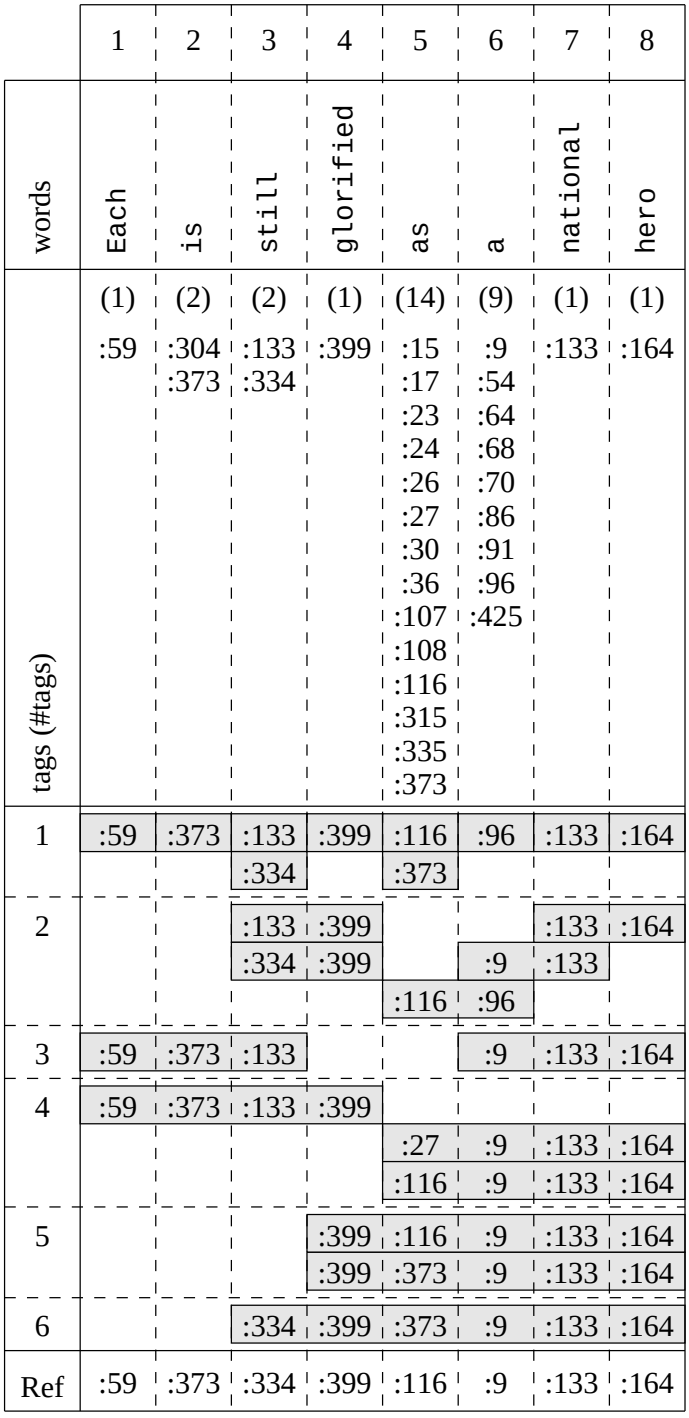


Figure 3.6: Reference tree for `trace_np_sample_2` (the 5-word sentence in figure 3.5)



	1	2	3	4	5	6	7	8
words	Each	is	still	glorified	as	a	national	hero
	(1)	(2)	(2)	(1)	(14)	(9)	(1)	(1)
tags (#tags)	:59	:304 :373	:133 :334	:399	:15 :17 :23 :24 :26 :27 :30 :36 :107 :108 :116 :315 :335 :373	:9 :54 :64 :68 :70 :86 :91 :96 :425	:133	:164
1	:59	:373	:133 :334	:399	:116 :373	:96	:133	:164
2			:133 :334	:399 :399		:9 :116	:133 :133 :96	:164 :164
3	:59	:373	:133			:9	:133	:164
4	:59	:373	:133	:399	:27 :116	:9 :9	:133 :133	:164 :164
5				:399 :399	:116 :373	:9 :9	:133 :133	:164 :164
6			:334	:399	:373	:9	:133	:164
Ref	:59	:373	:334	:399	:116	:9	:133	:164

Figure 3.7: Forest for `trace_np_sample_3` (a 8-word sentence with a cross product of 504 possible tags)

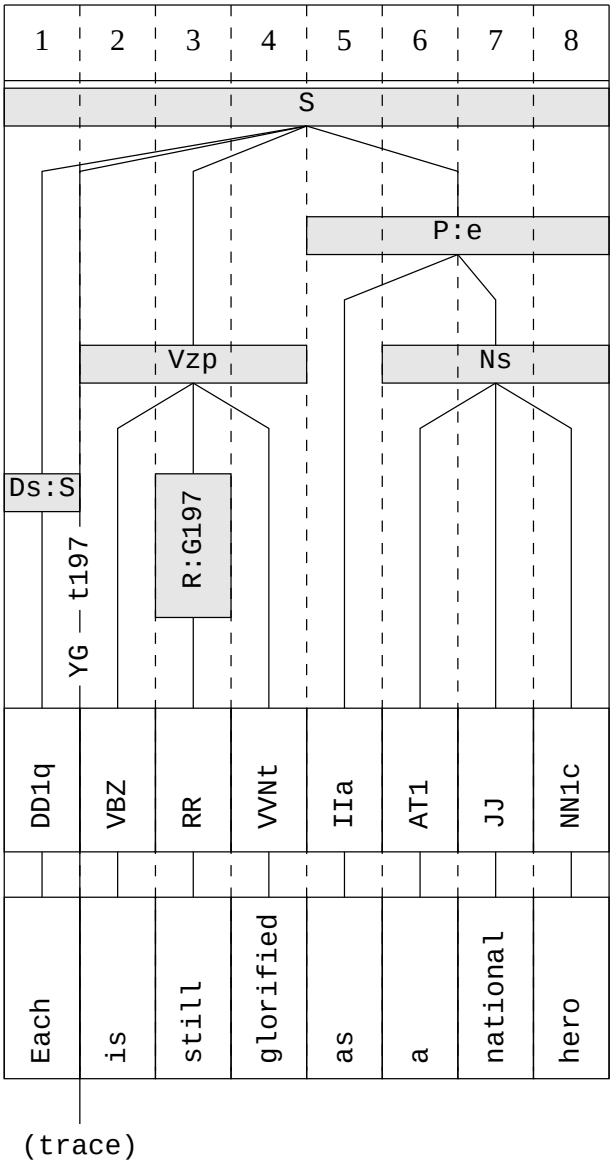
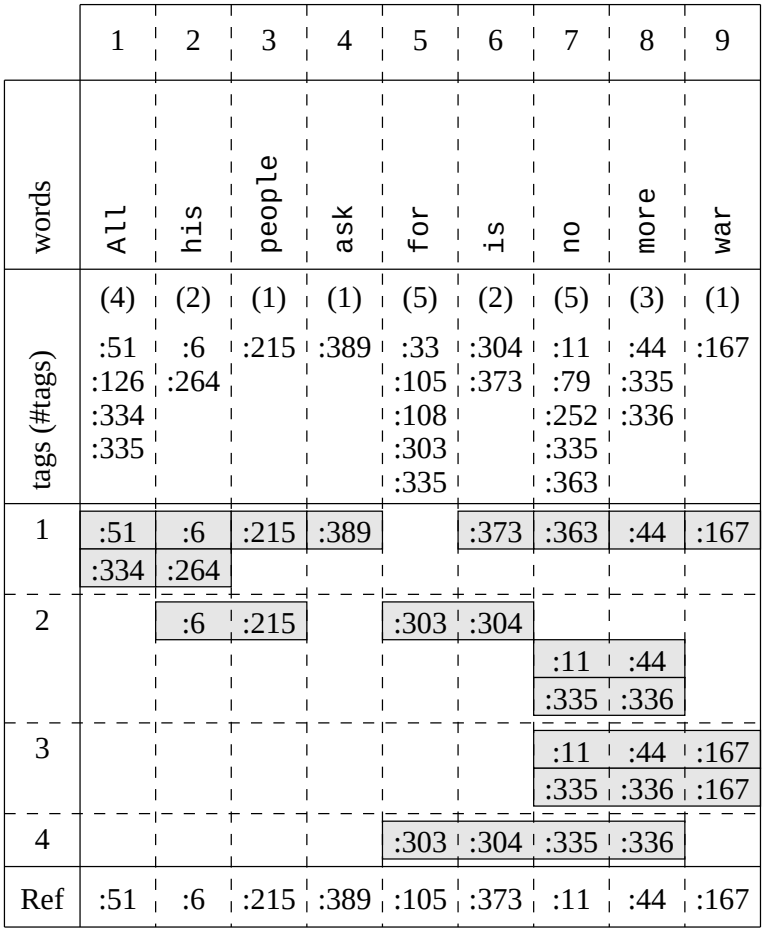


Figure 3.8: Reference tree for `trace_np_sample_3` (the 8-word sentence in figure 3.7)



	1	2	3	4	5	6	7	8	9
words	All	his	people	ask	for	is	no	more	war
tags (#tags)	(4) :51 :126 :334 :335	(2) :6 :264	(1) :215	(1) :389	(5) :33 :105 :108 :303 :335	(2) :304 :373	(5) :11 :79 :252 :335 :363	(3) :44 :335 :336	(1) :167
1	:51 :334	:6 :264	:215	:389		:373	:363	:44	:167
2		:6	:215		:303	:304			
							:11 :335	:44 :336	
3							:11 :335	:44 :336	:167 :167
4					:303	:304	:335	:336	
Ref	:51	:6	:215	:389	:105	:373	:11	:44	:167

Figure 3.9: Forest for `trace_np_sample_4` (a 9-word sentence with a cross product of 1,200 possible tags)

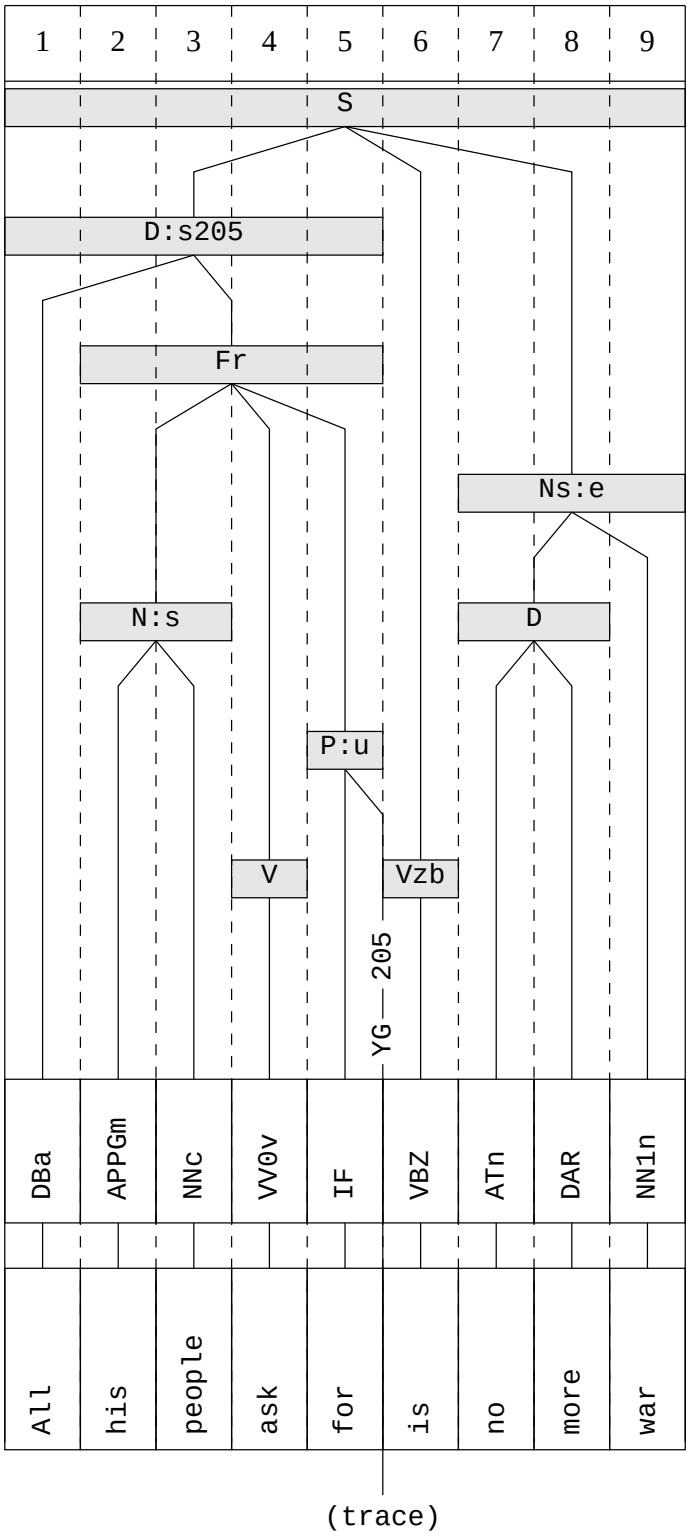
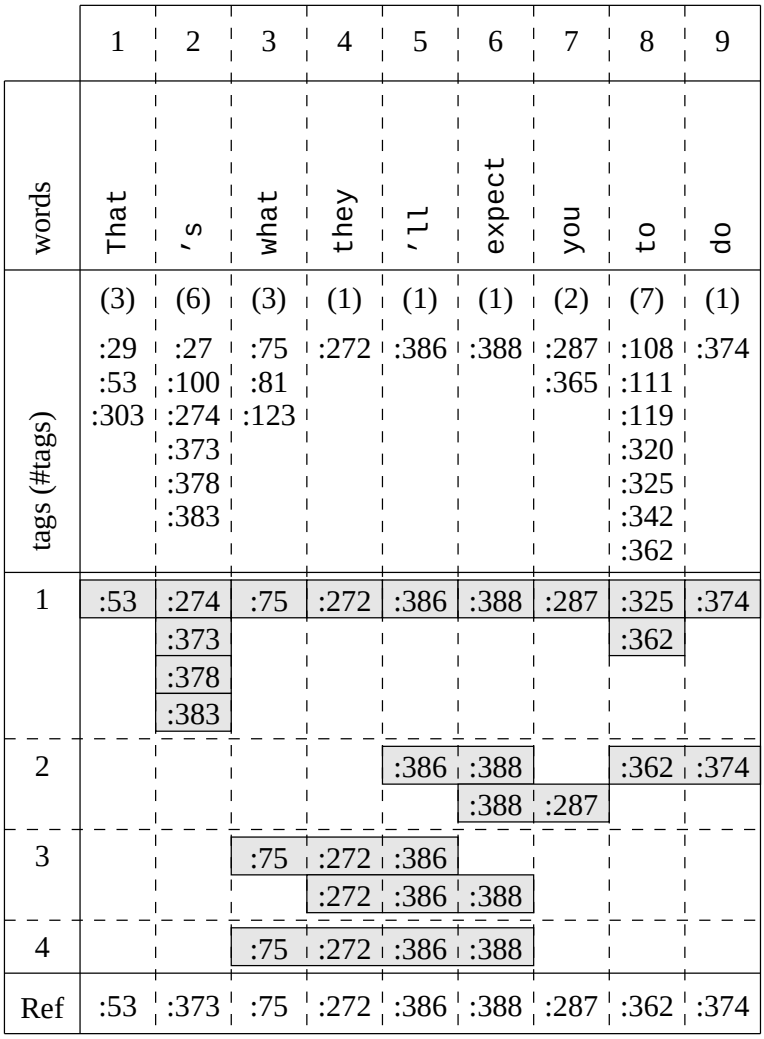


Figure 3.10: Reference tree for `trace_np_sample_4` (the 9-word sentence in figure 3.9)



	1	2	3	4	5	6	7	8	9
words	That	’s	what	they	’ll	expect	you	to	do
tags (#tags)	(3) :29 :53 :303	(6) :27 :100 :274 :373 :378 :383	(3) :75 :81 :123	(1) :272	(1) :386	(1) :388	(2) :287 :365	(7) :108 :111 :119 :320 :325 :342 :362	(1) :374
1	:53	:274 :373 :378 :383	:75	:272	:386	:388	:287	:325 :362	:374
2					:386	:388 :388	 :287	:362	:374
3			:75	:272 :272	:386 :386	 :388			
4			:75	:272	:386	:388			
Ref	:53	:373	:75	:272	:386	:388	:287	:362	:374

Figure 3.11: Forest for `trace_np_sample_5` (a 9-word sentence with a cross product of 756 possible tags)

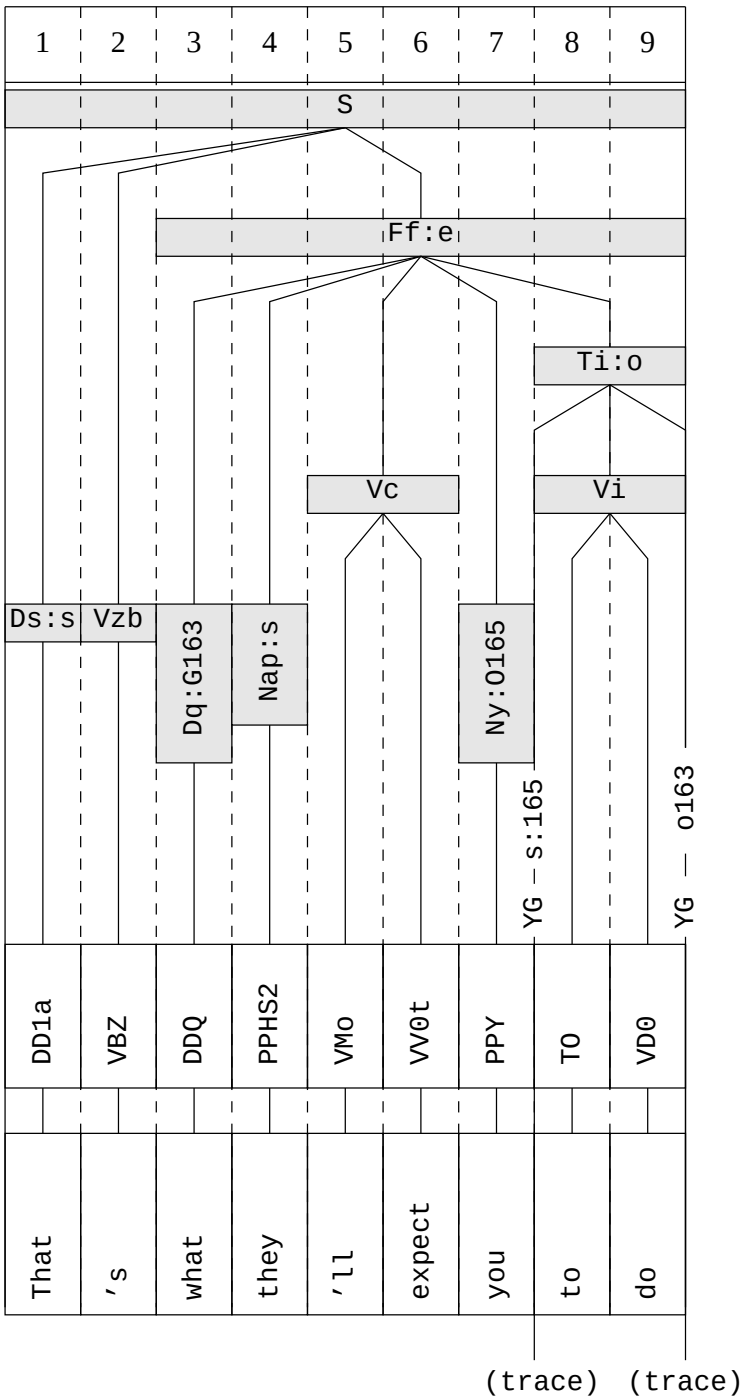


Figure 3.12: Reference tree for `trace_np_sample_5` (the 9-word sentence in figure 3.11)

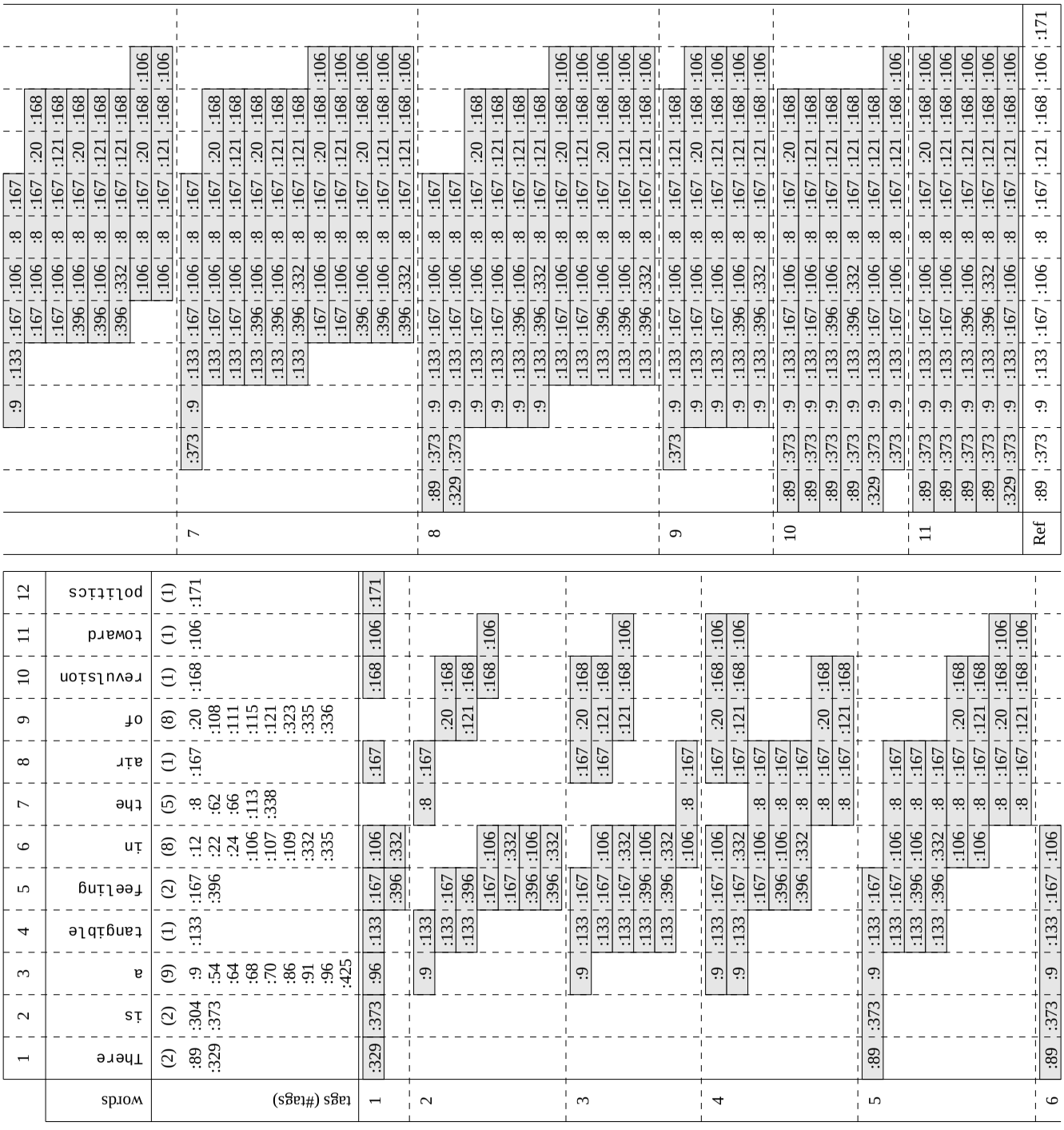


Figure 3.13: Forest for `trace_np_sample_6` (a 12-word sentence with a cross product of 23,040 possible tags)

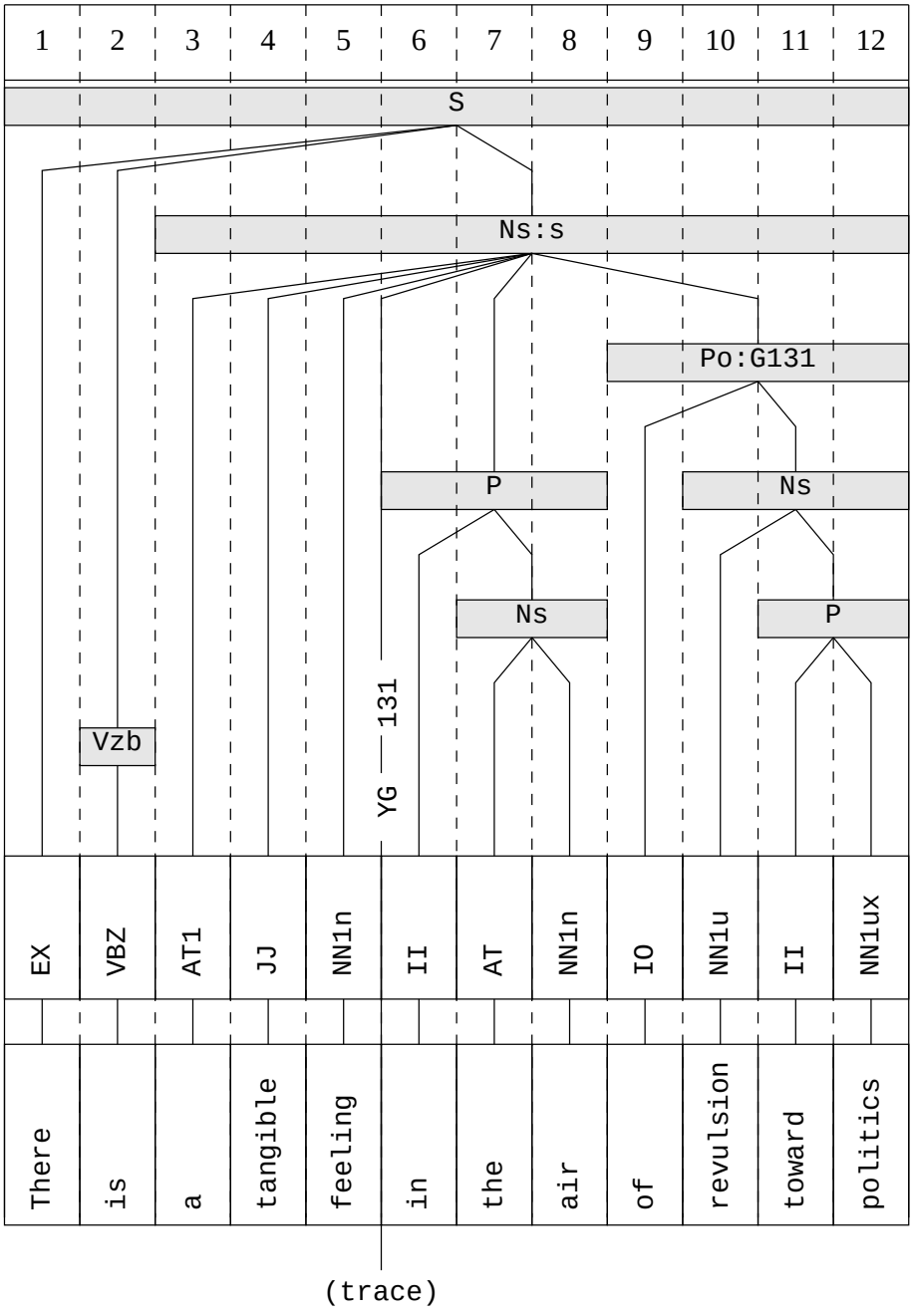


Figure 3.14: Reference tree for `trace_np_sample_6` (the 12-word sentence in figure 3.13)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
words	The	mystique	of	sex	,	combined	with	marijuana	and	jazz	,	is	intended	to	provide	a	design	for	living
tags (#tags)	(2)	(1)	(8)	(1)	(1)	(2)	(5)	(1)	(4)	(1)	(1)	(2)	(2)	(7)	(1)	(9)	(1)	(5)	(3)
	:8 :66	:164	:20 :108 :111 :115 :121 :323 :335 :336	:167	:408	:392 :400	:108 :109 :111 :122 :124	:168	:14 :61 :292 :338	:168	:408	:304 :373	:391 :399	:108 :111 :119 :320 :325 :342 :362	:389	:9 :54 :64 :68 :70 :86 :91 :96 :425	:167	:33 :105 :108 :303 :335	:128 :167 :396
1		:164		:167		:392 :400		:168	:14	:168		:373	:391 :399	:325 :362	:389	:96	:167		:128 :167 :396
2	:8	:164					:122	:168				:373	:399		:389	:96		:105	:167
			:20	:167					:14	:168			:391	:325				:105	:396
			:121	:167									:391	:362		:9	:167		
													:399	:325					
														:362	:389				
3		:164	:20	:167		:392	:122	:168					:391	:362	:389		:167	:105	:167
		:164	:121	:167		:400	:122	:168					:399	:362	:389		:167	:105	:396
				:167	:408	:400		:168	:14	:168				:362	:389	:96			
															:389	:9	:167		
4	:8	:164	:20	:167			:122	:168	:14	:168			:391	:362	:389	:96			
	:8	:164	:121	:167									:399	:362	:389	:96			
			:121	:167	:408	:400								:362	:389	:9	:167		
																:9	:167	:105	:167
																:9	:167	:105	:396
5		:164	:121	:167	:408	:400							:391	:362	:389	:9	:167		
				:167	:408	:400	:122	:168					:399	:362	:389	:9	:167		
						:392	:122	:168	:14	:168					:389	:9	:167	:105	:167
						:400	:122	:168	:14	:168					:389	:9	:167	:105	:396
6	:8	:164	:121	:167	:408	:400								:362	:389	:9	:167	:105	:167
			:121	:167	:408	:400	:122	:168						:362	:389	:9	:167	:105	:396
7		:164	:121	:167	:408	:400	:122	:168					:391	:362	:389	:9	:167	:105	:167
				:167	:408	:400	:122	:168	:14	:168			:391	:362	:389	:9	:167	:105	:396
													:399	:362	:389	:9	:167	:105	:167
													:399	:362	:389	:9	:167	:105	:396
8	:8	:164	:121	:167	:408	:400	:122	:168											
			:121	:167	:408	:400	:122	:168	:14	:168									
9		:164	:121	:167	:408	:400	:122	:168	:14	:168									
10	:8	:164	:121	:167	:408	:400	:122	:168	:14	:168									
Ref	:8	:164	:121	:167	:408	:400	:122	:168	:14	:168	:408	:373	:399	:362	:389	:9	:167	:105	:396

Figure 3.15: Forest for `trace_np_sample_7` (a 19-word sentence with a cross product of 2,419,200 possible tags)

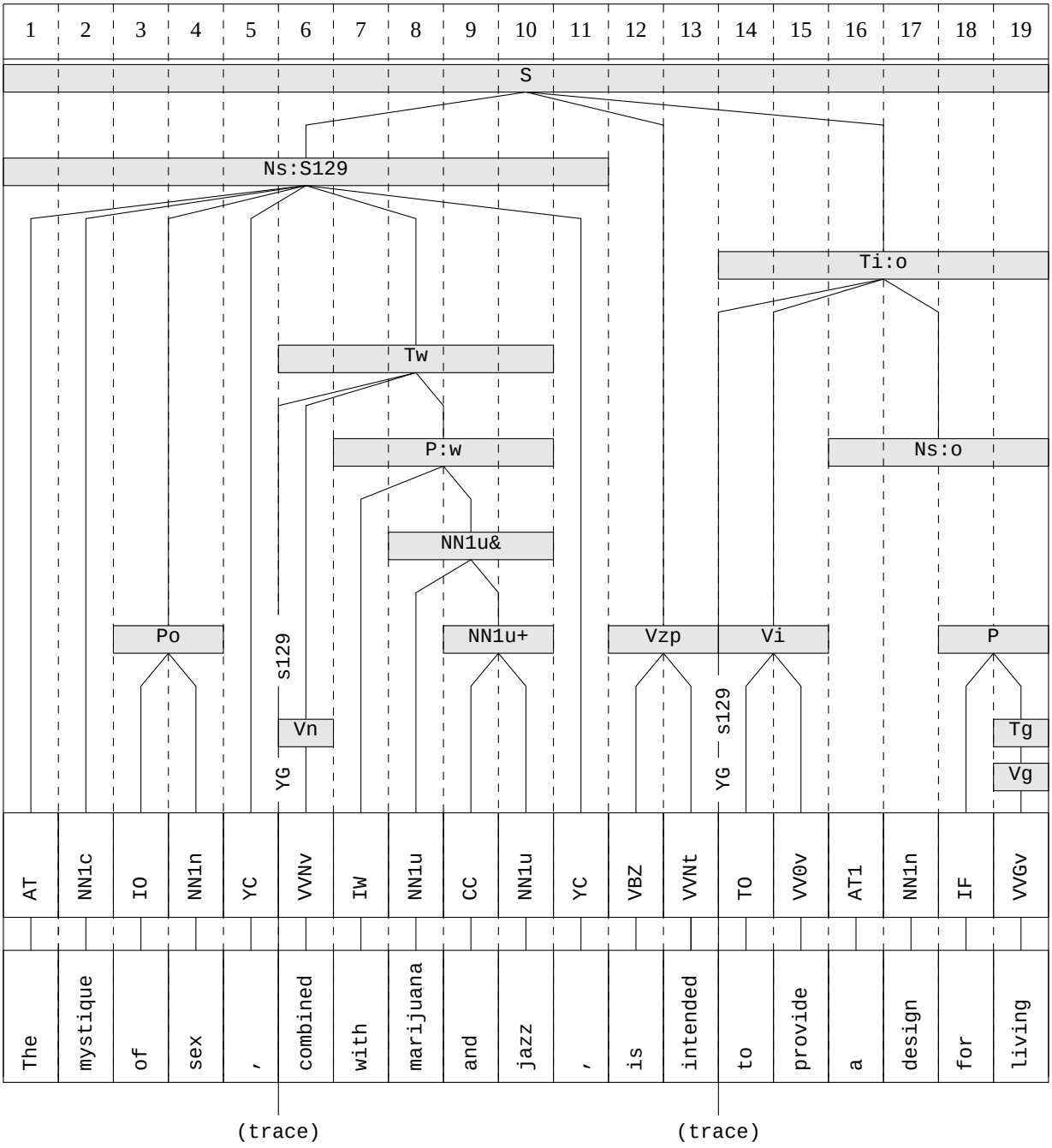


Figure 3.16: Reference tree for `trace_np_sample_7` (the 19-word sentence in figure 3.15)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
words	"	I	'd	just	turned	on	the	ignition	when	there	was	a	big	flash	and	I	was	lying	on	the	driveway	"	,	he	said
	(1)	(4)	(2)	(2)	(2)	(6)	(5)	(1)	(3)	(2)	(1)	(9)	(1)	(1)	(4)	(4)	(1)	(1)	(6)	(5)	(1)	(1)	(1)	(1)	(2)
tags (#tags)	:414	:97 :156 :234 :275	:380 :385	:133 :334	:392 :400	:106 :107 :109 :112 :294 :332	:8 :62 :66 :113 :338	:168	:38 :348 :349	:89 :329	:368	:9 :54 :64 :68 :70 :86 :91 :96 :425	:133	:164	:14 :61 :292 :338	:97 :156 :234 :275	:368	:394	:106 :107 :109 :112 :294 :332	:8 :62 :66 :113 :338	:164	:415	:408	:271	:392 :400
1		:275	:380 :385	:133 :334	:392 :400	:106 :332		:168	:38 :348 :349	:329	:368	:96	:133	:164	:14	:275	:368	:394	:106 :332		:164			:271	:392 :400
2			:380	:334	:133 :334 :392 :392 :400 :400	:400 :400 :106 :332 :106 :332	:8	:168				:9	:133 :133	:164	:14 :14	:156 :275	:368	:394 :394 :394	:106 :332	:8	:164			:271	:392
3		:275	:380	:133 :334	:400 :400	:106 :106	:8	:168		:89	:368	:96 :9	:133	:164					:106	:8	:164				
4		:275	:380	:133	:400 :392 :392 :400	:106 :106 :332 :106	:8 :8 :8 :8	:168 :168 :168 :168									:275	:368	:394	:106 :394 :106 :394 :332	:8 :8 :8	:164 :164 :164				
5				:334	:400	:106	:8	:168		:89	:368	:9	:133	:164							:164	:415	:408	:271	:392
6																:275	:368	:394	:106	:8	:164				
7																			:106	:8 :8	:164 :164	:415 :415	:408 :408	:271 :271	:392 :392
8																		:394	:106	:8	:164	:415	:408	:271	:392
10																:275	:368	:394	:106	:8	:164	:415	:408	:271	:392
Ref	:414	:275	:380	:334	:400	:332	:8	:168	:38	:89	:368	:9	:133	:164	:14	:275	:368	:394	:106	:8	:164	:415	:408	:271	:392

Figure 3.17: Forest for `trace_np_sample_8` (a 25-word sentence with a cross product of 49,766,400 possible tags)

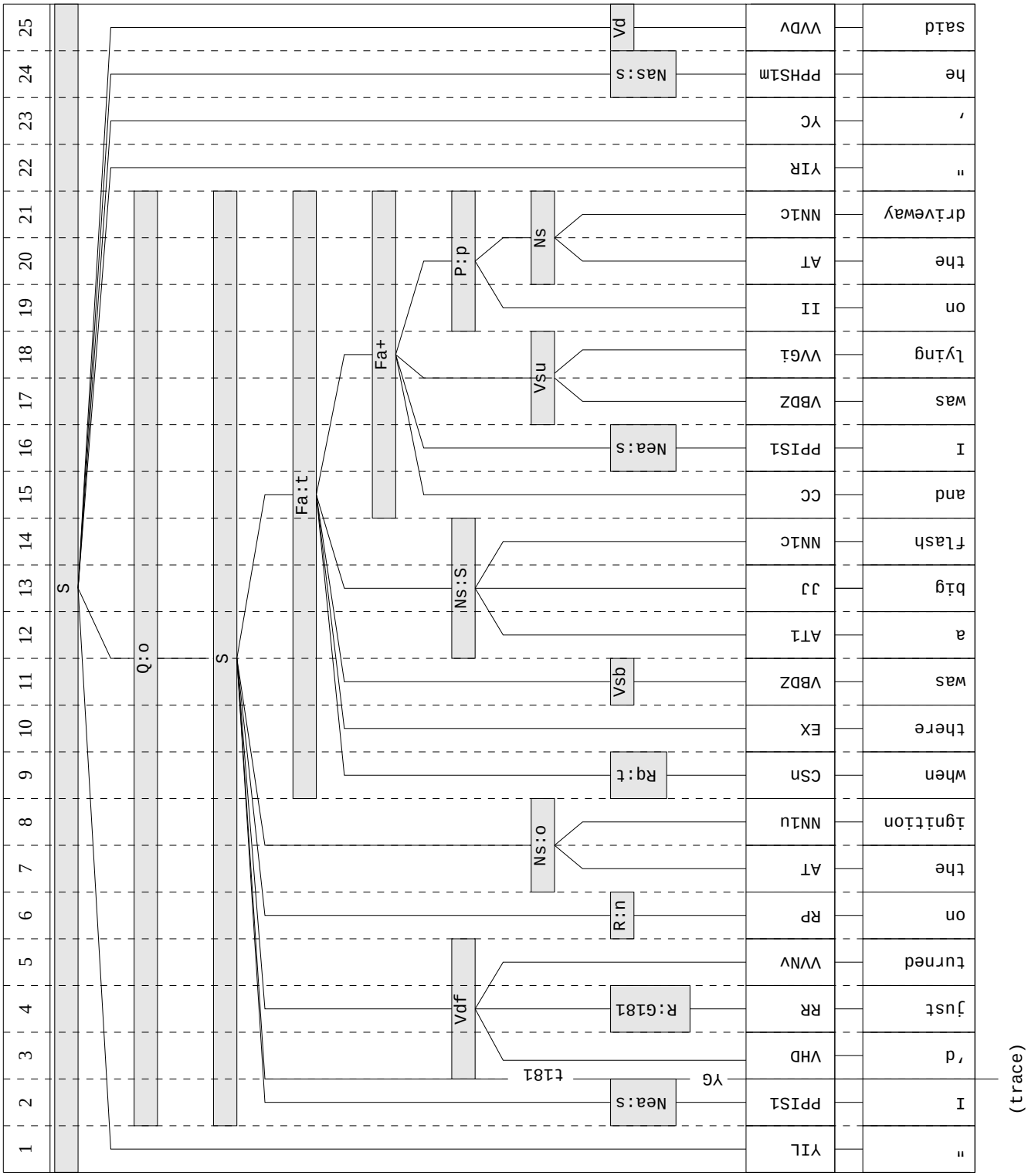


Figure 3.18: Reference tree for `trace_np_sample_8` (the 25-word sentence in figure 3.17)

Chapter 4

Design of strategies to combine syntactic constraints

Let us consider the cardinal of the cross product of the tags of the words, for a given sentence. This number is the total number of possible taggings for each sentence, and as we can see in the data of section C.5 and in the figure 4.1, this value can be very high throughout the `Trace` corpus.

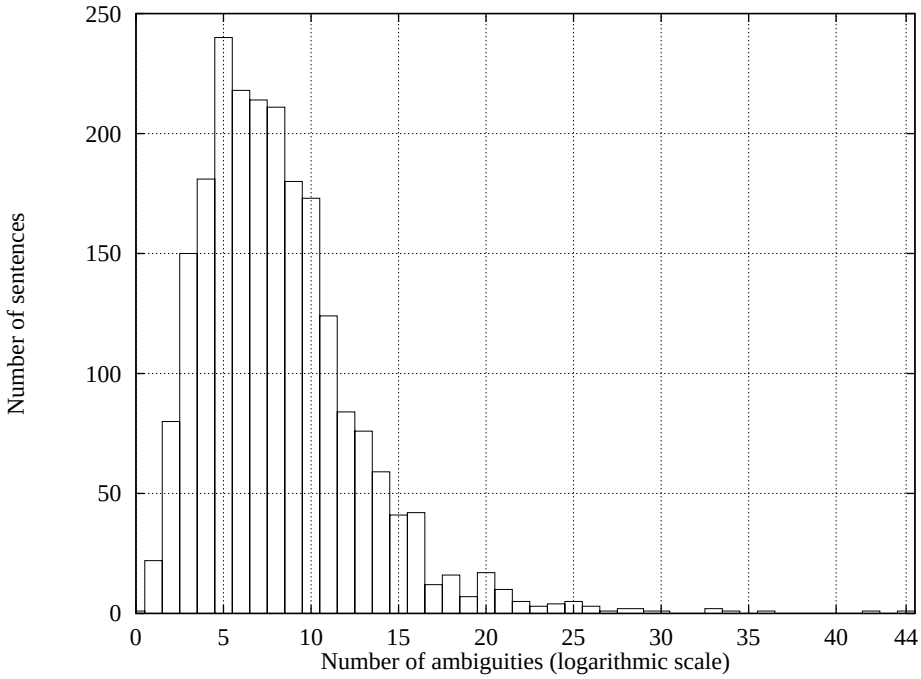


Figure 4.1: Cross products on `Trace` corpus

Therefore, our main objective now is to use the syntactic constraints introduced in the previous chapter to filter most of the possibilities, and reduce the cross product of the tags to a very low number, for instance between 20 and 50. After this reduction, the probability for each one of these sequences of tags could be estimated for instance by a forward-backward evaluation algorithm [6] applied to a hidden Markov model previously trained on the `Notrace` corpus. In this way, we would be able to establish an order in this list of sequences and choose one or more sequences from among the most probable ones.

In order to perform this reduction, we have implemented two different strategies to combine the subsequences of tags. Both are very simple and are perfectly described in the following sections. But

before this, as we did in the previous chapters, we have retagged `Trace_np_25` and `Trace_np_50` corpora with the BRILL system, trained on the `Notrace` corpus and using the `SUSANNE_full_LEXICON`, in order to have concrete numbers to compare with the performance of our strategies. The performance of BRILL on these two corpora is shown in table 4.1.

BRILL (full lexicon)			
	`Trace_np_25`	`Trace_np_50`	Total
#F	18071	32934	51005
OOVF+	0	0	0
OOVF-	0	0	0
NAF+	9545	17111	26656
NAF-	0	0	0
AF+	7603	14379	21982
AF-	923	1444	2367
S1	94.89	95.62	95.36
S2	89.17	90.87	90.28

Table 4.1: BRILL system tagging `Trace_np_25` and `Trace_np_50` corpora

4.1 Strategy 1: take the longest and most probable sequence of tags

Our first strategy starts by taking the longest sequence of tags, and then filling the holes recursively, by using the same criterion, until the sequence covers the whole sentence. When we find more than one sequence with the same length we always take the most probable one. This deterministic behaviour makes this strategy produce only one new sequence of tags for each sentence. This first technique can be applied to both `Trace_np_25` and `Trace_np_50` corpora by the following commands:

```
cat SUSANNE_trace_np_25_FOREST | strategy_1.prl > results/results_stgy_1_trace_np_25
cat SUSANNE_trace_np_50_FOREST | strategy_1.prl > results/results_stgy_1_trace_np_50
```

As an example, here we have the sequences produced by `strategy_1.prl` for the sentences `trace_np_sample_1` to `trace_np_sample_8`, together with the sequence present in the reference corpus. The first number immediately followed by `x` is the number of times this sequence appears (always 1 here), and the last number in parenthesis is the number of errors found when we compare the sequence with the reference.

```
trace_np_sample_1 (4 words, 0 errors):
 1x :389 :266 :389 :332 (0)
Ref :389 :266 :389 :332

trace_np_sample_2 (5 words, 0 errors):
 1x :53 :373 :51 :275 :389 (0)
Ref :53 :373 :51 :275 :389

trace_np_sample_3 (8 words, 1 errors):
 1x :59 :373 :334 :399 :373 :9 :133 :164 (1)
Ref :59 :373 :334 :399 :116 :9 :133 :164

trace_np_sample_4 (9 words, 5 errors):
 1x :126 :6 :215 :389 :303 :304 :335 :336 :167 (5)
Ref :51  :6 :215 :389 :105 :373 :11  :44  :167

trace_np_sample_5 (9 words, 2 errors):
 1x :53 :100 :75 :272 :386 :388 :365 :362 :374 (2)
Ref :53 :373 :75 :272 :386 :388 :287 :362 :374

trace_np_sample_6 (12 words, 0 errors):
 1x :89 :373 :9 :133 :167 :106 :8 :167 :121 :168 :106 :171 (0)
Ref :89 :373 :9 :133 :167 :106 :8 :167 :121 :168 :106 :171

trace_np_sample_7 (19 words, 0 errors):
```

```
 1x :8 :164 :121 :167 :408 :400 :122 :168 :14 :168 :408 :373 :399 :362 :389 :9 :167 :105 :396 (0)
Ref :8 :164 :121 :167 :408 :400 :122 :168 :14 :168 :408 :373 :399 :362 :389 :9 :167 :105 :396

trace_np_sample_8 (25 words, 2 errors):
 1x ... :400 :106 :8 :168 :38 :89 :368 :9 :133 :164 :292 :275 :368 :394 :106 :8 :164 :415 :408 :271 :392 (2)
Ref ... :400 :332 :8 :168 :38 :89 :368 :9 :133 :164 :14  :275 :368 :394 :106 :8 :164 :415 :408 :271 :392
```

Finally, table 4.2 shows the results of the first strategy for all the sentences, and we can see that the performance is lower than the performance of BRILL tagger. The reason could be that we have implemented a very dramatic reduction (from a very high number of possibilities to only one), but another concept involved here is that perhaps the longest trees are not always the best candidates to choose for correctly disambiguating the sentence, because they could imply very strong constraints, and perhaps it is a better idea to use a less deterministic starting point.

Strategy 1			
	Trace_np_25	Trace_np_50	Total
#F	18071	32934	51005
OOVF+	0	0	0
OOVF-	0	0	0
NAF+	9545	17111	26656
NAF-	0	0	0
AF+	7055	13249	20304
AF-	1471	2574	4045
S1	91.86	92.18	92.07
S2	82.75	83.73	83.39

Table 4.2: Strategy 1 tagging Trace_np_25 and Trace_np_50 corpora

4.2 Strategy 2: the longest and most probable, but once per cell

Due to the reasons discussed above, we have implemented a second strategy of combination. As before, we move all the information in the forest to a CYK table and recursively build several sequences of tags. The criterion is again to always take the longest and most probable tree, but instead of starting at the top of the CYK table, we apply the same procedure to each cell. Therefore, the lowest number of sequences is 1, and the highest is

$$\frac{n \times (n+1)}{2}$$

where n is the number of words in the sentence. This second technique can be applied to both Trace_np_25 and Trace_np_50 corpora by the following commands:

```
cat SUSANNE_trace_np_25_FOREST | strategy_2.prl > results/results_stgy_2_trace_np_25
cat SUSANNE_trace_np_50_FOREST | strategy_2.prl > results/results_stgy_2_trace_np_50
```

Here we have the sequences produced by strategy_2.prl for the sentences trace_np_sample_1 to trace_np_sample_8, together with the sequence present in the reference corpus. As before, the first number immediately followed by x is the number of times this sequence appears, and the last number in parenthesis is the number of errors found when we compare the sequence with the reference. The stochastic module cited above to estimate the most probable sequence of tags among this small set is not available yet. Instead of that, we choose the sequence with the lowest number of errors directly. Therefore, the results shown here must be taken as an upper limit of the possible performance of this second technique.

```
trace_np_sample_1 (4 words, 6 sequences, 0 errors):
 6x :389 :266 :389 :332 (0)
Ref :389 :266 :389 :332

trace_np_sample_2 (5 words, 8 sequences, 0 errors):
```

```
 2x :53 :100 :51  :275 :389 (1)
 2x :53 :27  :334 :275 :389 (2)
 4x :53 :373 :51  :275 :389 (0)
Ref :53 :373 :51  :275 :389

trace_np_sample_3 (8 words, 18 sequences, 0 errors):
 5x :59 :373 :133 :399 :116 :9  :133 :164 (1)
 3x :59 :373 :133 :399 :116 :96 :133 :164 (2)
 1x :59 :373 :133 :399 :15  :54 :133 :164 (3)
 4x :59 :373 :133 :399 :15  :9  :133 :164 (2)
 2x :59 :373 :334 :399 :116 :9  :133 :164 (0)
 3x :59 :373 :334 :399 :373 :9  :133 :164 (1)
Ref :59 :373 :334 :399 :116 :9  :133 :164

trace_np_sample_4 (9 words, 14 sequences, 1 error):
 2x :126 :264 :215 :389 :303 :304 :335 :336 :167 (6)
 2x :126 :6   :215 :389 :105 :373 :11  :44  :167 (1)
 3x :126 :6   :215 :389 :303 :304 :11  :44  :167 (3)
 5x :126 :6   :215 :389 :303 :304 :335 :336 :167 (5)
 2x :126 :6   :215 :389 :303 :304 :79  :44  :167 (4)
Ref :51  :6   :215 :389 :105 :373 :11  :44  :167

trace_np_sample_5 (9 words, 15 sequences, 1 error):
 2x :53 :100 :123 :272 :386 :388 :287 :362 :374 (2)
 3x :53 :100 :123 :272 :386 :388 :365 :362 :374 (3)
 2x :53 :100 :75  :272 :386 :388 :287 :362 :374 (1)
 2x :53 :100 :75  :272 :386 :388 :365 :342 :374 (3)
 6x :53 :100 :75  :272 :386 :388 :365 :362 :374 (2)
Ref :53 :373 :75  :272 :386 :388 :287 :362 :374

trace_np_sample_6 (12 words, 46 sequences, 0 errors):
 5x :89 :373 :54 :133 :167 :106 :8   :167 :121 :168 :106 :171 (1)
 1x :89 :373 :54 :133 :167 :332 :8   :167 :121 :168 :106 :171 (2)
 1x :89 :373 :54 :133 :396 :106 :8   :167 :121 :168 :106 :171 (2)
 4x :89 :373 :9  :133 :167 :106 :113 :167 :121 :168 :106 :171 (1)
 3x :89 :373 :9  :133 :167 :106 :8   :167 :115 :168 :106 :171 (1)
25x :89 :373 :9  :133 :167 :106 :8   :167 :121 :168 :106 :171 (0)
 1x :89 :373 :9  :133 :167 :12  :8   :167 :121 :168 :106 :171 (1)
 1x :89 :373 :9  :133 :167 :332 :8   :167 :121 :168 :106 :171 (1)
 4x :89 :373 :9  :133 :396 :106 :8   :167 :121 :168 :106 :171 (1)
 1x :89 :373 :9  :133 :396 :332 :8   :167 :121 :168 :106 :171 (2)
Ref :89 :373 :9  :133 :167 :106 :8   :167 :121 :168 :106 :171

trace_np_sample_7 (19 words, 58 sequences, 0 errors):
 6x :66 :164 :121 :167 :408 :400 :122 :168 :14  :168 :408 :373 :399 :362 :389 :9  :167 :105 :396 (1)
 5x :8  :164 :115 :167 :408 :400 :122 :168 :14  :168 :408 :373 :399 :362 :389 :9  :167 :105 :396 (1)
 1x :8  :164 :121 :167 :408 :400 :122 :168 :14  :168 :408 :373 :391 :362 :389 :9  :167 :105 :396 (1)
 1x :8  :164 :121 :167 :408 :400 :122 :168 :14  :168 :408 :373 :399 :325 :389 :9  :167 :105 :396 (1)
 4x :8  :164 :121 :167 :408 :400 :122 :168 :14  :168 :408 :373 :399 :342 :389 :9  :167 :105 :396 (1)
 1x :8  :164 :121 :167 :408 :400 :122 :168 :14  :168 :408 :373 :399 :342 :389 :96 :167 :105 :396 (2)
 1x :8  :164 :121 :167 :408 :400 :122 :168 :14  :168 :408 :373 :399 :362 :389 :54 :167 :105 :396 (1)
 2x :8  :164 :121 :167 :408 :400 :122 :168 :14  :168 :408 :373 :399 :362 :389 :9  :167 :105 :128 (1)
28x :8  :164 :121 :167 :408 :400 :122 :168 :14  :168 :408 :373 :399 :362 :389 :9  :167 :105 :396 (0)
 4x :8  :164 :121 :167 :408 :400 :122 :168 :14  :168 :408 :373 :399 :362 :389 :96 :167 :105 :396 (1)
 2x :8  :164 :121 :167 :408 :400 :122 :168 :292 :168 :408 :373 :399 :362 :389 :9  :167 :105 :396 (1)
 3x :8  :164 :121 :167 :408 :400 :124 :168 :14  :168 :408 :373 :399 :362 :389 :9  :167 :105 :396 (1)
Ref :8  :164 :121 :167 :408 :400 :122 :168 :14  :168 :408 :373 :399 :362 :389 :9  :167 :105 :396

trace_np_sample_8 (25 words, 54 sequences, 1 error):
 3x ... :392 :106 :8   :168 :38 :89 :368 :9  :133 :164 :292 :275 :368 :394 :106 :8   :164 :415 :408 :271 :392 (4)
 2x ... :400 :106 :8   :168 :38 :89 :368 :9  :133 :164 :292 :275 :368 :394 :106 :8   :164 :415 :408 :271 :392 (3)
 2x ... :400 :112 :113 :168 :38 :89 :368 :9  :133 :164 :292 :275 :368 :394 :106 :8   :164 :415 :408 :271 :392 (4)
 2x ... :400 :112 :8   :168 :38 :89 :368 :9  :133 :164 :292 :275 :368 :394 :106 :8   :164 :415 :408 :271 :392 (3)
 1x ... :400 :332 :8   :168 :38 :89 :368 :9  :133 :164 :292 :275 :368 :394 :106 :8   :164 :415 :408 :271 :392 (2)
 2x ... :392 :106 :8   :168 :38 :89 :368 :9  :133 :164 :292 :275 :368 :394 :106 :8   :164 :415 :408 :271 :392 (3)
 1x ... :400 :106 :8   :168 :38 :89 :368 :54 :133 :164 :292 :275 :368 :394 :106 :8   :164 :415 :408 :271 :392 (3)
 8x ... :400 :106 :8   :168 :38 :89 :368 :9  :133 :164 :14  :275 :368 :394 :106 :8   :164 :415 :408 :271 :392 (1)
 1x ... :400 :106 :8   :168 :38 :89 :368 :9  :133 :164 :14  :275 :368 :394 :112 :8   :164 :415 :408 :271 :392 (2)
 1x ... :400 :106 :8   :168 :38 :89 :368 :9  :133 :164 :14  :275 :368 :394 :332 :8   :164 :415 :408 :271 :392 (2)
 3x ... :400 :106 :8   :168 :38 :89 :368 :9  :133 :164 :292 :275 :368 :394 :106 :113 :164 :415 :408 :271 :392 (3)
24x ... :400 :106 :8   :168 :38 :89 :368 :9  :133 :164 :292 :275 :368 :394 :106 :8   :164 :415 :408 :271 :392 (2)
 4x ... :400 :106 :8   :168 :38 :89 :368 :96 :133 :164 :292 :275 :368 :394 :106 :8   :164 :415 :408 :271 :392 (3)
Ref ... :400 :332 :8   :168 :38 :89 :368 :9  :133 :164 :14  :275 :368 :394 :106 :8   :164 :415 :408 :271 :392
```

Table 4.3 shows the results of the second strategy for all the sentences, and now we can see that the performance is somewhat higher than the performance of BRILL tagger. Of course, we need to implement the estimation algorithm, design more sophisticated strategies of combination, and extend all these techniques to deal with out-of-vocabulary forms, but these results seem to point in the right direction.

Strategy 2			
	`Trace_np_25`	`Trace_np_50`	Total
#F	18071	32934	51005
OOVF+	0	0	0
OOVF-	0	0	0
NAF+	9545	17111	26656
NAF-	0	0	0
AF+	7878	14399	22277
AF-	648	1424	2072
S1	96.41	95.68	95.94
S2	92.40	91.00	91.49

Table 4.3: Strategy 2 tagging `Trace_np_25` and `Trace_np_50` corpora

Chapter 5

Conclusion

The present work is not over yet. We have seen that the use of the syntactic constraints imposed by a grammar can be very useful for disambiguating a text in natural language. However, there are still a lot of interesting experiments to be performed, all of them with the purpose of bringing the conditions under which this study has been done nearer and nearer to what happens in real life.

For instance, just as it is true that in practice it is very strange to have a grammar available that can parse all of the sentences of a given language, it is also true that is very strange to have complete lexica available. Therefore, all of the results shown here must be taken as upper limits for the performance of these kinds of techniques.

For this reason, our work in the immediate future will focus on the implementation of a tool that is more directed to the task of disambiguation than to parsing, and able to:

- obtain the forest of subtrees of a given sentence, even when there are out-of-vocabulary forms, and with a more reasonable computational time than the ones shown here,
- integrate new and more sophisticated strategies to combine those subtrees with an improved performance in the disambiguation process,
- and also provide a stochastic module to estimate correctly which is the most probable sequence of tags for those combination strategies that reduce the cross product of tags to more than one possibility of tagging.

Finally, we would like also to point out that a grammar is a very hard linguistic resource to design, and may be under-used if we take advantage of it only to disambiguate texts. Therefore, another future subject of research is of course to design strategies that will automatically extract syntactic rules from the forests of non-complete subtrees, which is precisely the aim of robust parsing techniques.

Bibliography

[1] J.-C. Chappelier, M. Rajman. A Practical Bottom-Up Algorithm for On-Line Parsing with Stochastic Context-Free Grammars. *Technical Report, Artificial Intelligence Laboratory, Swiss Federal Institute of Technology, Lausanne.*

[2] G. Sampson. The Susanne Corpus, Release 3, 04/04/1994. *School of Cognitive & Computing Sciences, University of Sussex, Falmer, Brighton, England.*

[3] G. Sampson. English for the Computer. *Oxford University Press*, 1994.

[4] R.G. Garside, G.N. Leech, G.R. Sampson, eds. The Computational Analysis of English. Longman, 1987.

[5] E. Brill. Some advances in rule-based part of speech tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, Wa., 1994.

[6] L. Rabiner, B.H. Juang. Fundamentals of speech recognition. Chapter 6, Theory and implementation of Hidden Markov Models. Prentice-Hall, 1993.

Appendix A

SUSANNE tag set

The following list is the tag set used in the SUSANNE corpus. It is simply the enumeration of the tags. A description of each tag can be obtained from [3] and [4]. The cardinal of the tag set is 425 tags.

1) APPGf	2) APPGh1	3) APPGh2	4) APPGi1	5) APPGi2
6) APPGm	7) APPGy	8) AT	9) AT1	10) AT1e
11) ATn	12) BTO21	13) BTO22	14) CC	15) CC31
16) CC32	17) CC33	18) CCB	19) CCn	20) CCr
21) CS	22) CS21	23) CS22	24) CS31	25) CS32
26) CS33	27) CSA	28) CSN	29) CST	30) CST21
31) CST22	32) CSW	33) CSf	34) CSg	35) CSi
36) CSk21	37) CSk22	38) CSn	39) CSr	40) DA1
41) DA2	42) DA2R	43) DA2q	44) DAR	45) DAT
46) DAg	47) DAr	48) DAy	49) DAz	50) DB2
51) DBa	52) DBh	53) DD1a	54) DD1b21	55) DD1b22
56) DD1e	57) DD1i	58) DD1n	59) DD1q	60) DD1q41
61) DD1q42	62) DD1q43	63) DD1q44	64) DD1t21	65) DD1t22
66) DD21	67) DD22	68) DD221	69) DD222	70) DD231
71) DD232	72) DD233	73) DD2a	74) DD2i	75) DDQ
76) DDQGq	77) DDQGr	78) DDQV	79) DDQV31	80) DDQV32
81) DDQV33	82) DDQq	83) DDQr	84) DDf	85) DDi
86) DDo21	87) DDo22	88) DDy	89) EX	90) FA
91) FB	92) FO	93) FOc	94) FOqx	95) FOs
96) FOx	97) FW	98) FWg	99) FWs	100) GG
101) ICS	102) ICSk	103) ICSt	104) ICSx	105) IF
106) II	107) II21	108) II22	109) II31	110) II32
111) II33	112) II41	113) II42	114) II43	115) II44
116) IIa	117) IIb	118) IIp	119) IIt	120) IIx
121) IO	122) IW	123) IW21	124) IW22	125) JA
126) JA21	127) JA22	128) JB	129) JBR	130) JBT
131) JBo	132) JBy	133) JJ	134) JJ21	135) JJ22
136) JJR	137) JJT	138) JJh	139) JJj	140) JJs
141) LE	142) LE21	143) LE22	144) LEe	145) LEn
146) MC	147) MC1	148) MC1n	149) MC2	150) MC2y
151) MCb	152) MCd	153) MCe	154) MCn	155) MCo
156) MCr	157) MCy	158) MD	159) MDn	160) MDo
161) MDt	162) MFn	163) ND1	164) NN1c	165) NN1c21
166) NN1c22	167) NN1n	168) NN1u	169) NN1u21	170) NN1u22
171) NN1ux	172) NN2	173) NN221	174) NN222	175) NNJ
176) NNJ1c	177) NNJ1n	178) NNJ2	179) NNL	180) NNL1c
181) NNL1cb	182) NNL1n	183) NNL2	184) NNOc	185) NNS
186) NNS1c	187) NNS1c21	188) NNS1c22	189) NNS1n	190) NNS2
191) NNS21	192) NNS22	193) NNSA	194) NNSS	195) NNSj

196) NNT1c 197) NNT1h 198) NNT1h21 199) NNT1h22 200) NNT1m
201) NNT2 202) NNU 203) NNU1c 204) NNU1n 205) NNU2
206) NNU21 207) NNU22 208) NNUb 209) NNUc 210) NNUp
211) NNUp21 212) NNUp22 213) NNa 214) NNb 215) NNc
216) NNm 217) NNmm 218) NNn 219) NNu 220) NNux
221) NP1c 222) NP1c21 223) NP1c22 224) NP1f 225) NP1g
226) NP1i 227) NP1j 228) NP1j31 229) NP1j32 230) NP1j33
231) NP1m 232) NP1p 233) NP1p21 234) NP1p22 235) NP1s
236) NP1t 237) NP1t21 238) NP1t22 239) NP1x 240) NP1z
241) NP2c 242) NP2g 243) NP2p 244) NP2s 245) NP2z
246) NPD1 247) NPD2 248) NPM 249) NPM1 250) PN
251) PN1 252) PN121 253) PN122 254) PN1o 255) PN1z
256) PNQOr 257) PNQSq 258) PNQSr 259) PNQVS 260) PPGf
261) PPGh2 262) PPGi1 263) PPGi2 264) PPGm 265) PPGy
266) PPH1 267) PPHO1f 268) PPHO1m 269) PPHO2 270) PPHS1f
271) PPHS1m 272) PPHS2 273) PPIO1 274) PPIO2 275) PPIS1
276) PPIS2 277) PPX1f 278) PPX1h 279) PPX1i 280) PPX1m
281) PPX1y 282) PPX2h 283) PPX2h21 284) PPX2h22 285) PPX2i
286) PPX2y 287) PPY 288) RAa 289) RAc 290) RAc21
291) RAc22 292) RAc31 293) RAc32 294) RAc33 295) RAe
296) RAh 297) RAj 298) RAn 299) RAp21 300) RAp22
301) RAq 302) REX 303) REX21 304) REX22 305) RG
306) RG21 307) RG22 308) RGA 309) RGAf 310) RGQV
311) RGQV31 312) RGQV32 313) RGQV33 314) RGQq 315) RGa
316) RGb 317) RGf 318) RGi 319) RGi21 320) RGi22
321) RGr 322) RGr21 323) RGr22 324) RGz 325) RL
326) RL21 327) RL22 328) RLe 329) RLh 330) RLn
331) RLw 332) RP 333) RPK 334) RR 335) RR21
336) RR22 337) RR31 338) RR32 339) RR33 340) RR41
341) RR42 342) RR43 343) RR44 344) RRQV 345) RRQV31
346) RRQV32 347) RRQV33 348) RRQq 349) RRQr 350) RRR
351) RRT 352) RRe 353) RRf 354) RRg 355) RRs
356) RRx 357) RRz 358) RT 359) RTn 360) RTo
361) RTt 362) TO 363) UH 364) UH21 365) UH22
366) VB0 367) VBDR 368) VBDZ 369) VBG 370) VBM
371) VBN 372) VBR 373) VBZ 374) VD0 375) VDD
376) VDG 377) VDN 378) VDZ 379) VH0 380) VHD
381) VHG 382) VHN 383) VHZ 384) VMK 385) VMd
386) VMo 387) VV0i 388) VV0t 389) VV0v 390) VVDi
391) VVDt 392) VVDv 393) VVGK 394) VVGi 395) VVGt
396) VVGv 397) VVNK 398) VVNi 399) VVNt 400) VVNv
401) VVZi 402) VVZt 403) VVZv 404) XX 405) YB
406) YBL 407) YBR 408) YC 409) YD 410) YE
411) YF 412) YG 413) YH 414) YIL 415) YIR
416) YN 417) YO 418) YPL 419) YPR 420) YQ
421) YS 422) YTL 423) YTR 424) YX 425) ZZ1

Appendix B

SUSANNE notation for non-terminal node labels

Non-terminal node labels in the SUSANNE scheme contain up to three types of information: a *formtag*, a *functiontag*, and an *index*, in that order. In a label containing a formtag and one or both of the other two elements, a colon separates the formtag from the other elements. A functiontag is always a single alphabetic character, and an index is a sequence of three digits; restrictions on valid combinations of elements within a node label mean that complex labels can always be unambiguously decomposed into their elements.

B.1 Ranks of constituent

Apart from nodes immediately dominating "ghost" elements, all nodes have labels including formtags, which identify the internal properties of the word or word-sequence dominated by the node. The shape of a parse-tree is defined in terms of a hierarchy of formtag ranks:

1. Wordrank formtags (begin with two capital letters; formtags of all other ranks begin with one capital and contain no further capitals).

2. Phrasetags (begin with one of: `N`, `V`, `J`, `R`, `P`, `D`, `M`, `G`).

3. Clausetags (begin with one of: `S`, `F`, `T`, `Z`, `L`, `A`, `W`).

4. Rootrank formtags (begin with one of: `O`, `Q`, `I`,).

Each grammatical clause, whether consisting of one or more words, is given a node labelled with a clausetag. Each immediate constituent of a clause, whether there are one or more such constituents and whether the constituent consists of one or more words, is given a node labelled with a phrasetag, unless the constituent belongs to a wordrank category that has no corresponding phraserank category (e.g. punctuation marks, conjunctions), or to a rootrank category (e.g. a direct quotation, formtagged `Q`). Thus a clause consisting of one verb will be assigned a clausetag (e.g. `Tg` for present-participle clause) which singularily dominates a phrasetag (e.g. `Vg` for verb group beginning with present participle) which in turn singularily dominates a wordrank formtag (e.g. `VVGi` for present participle of intransitive verb).

An intermediate phrase node is inserted between a higher phrase node and a sequence of words dominated by it only if two or more of those words form a coherent constituent within the higher phrase. A clause which fills a slot standardly filled by a phrase (e.g. a nominal clause as subject or object) will not have a phrase node above the clause node unless the clause proper is preceded and/or followed by modifying elements that are not part of the clause.

B.2 Functiontags and indices

Functiontags, identifying roles such as surface subject, logical object, time adjunct, are assigned to all immediate constituents of clauses, except for their verb-group heads and certain other constituents for which function labelling is inappropriate.

Indices are assigned to pairs of nodes to show referential identity between items which are in certain defined grammatical relationships to one another. For instance, a phrase raised out of a lower clause to act as object in a higher clause, as in `John expected Mary to admit it`, will be assigned an index identical to that assigned to the ghost element which marks the logical position of the item in the lower clause. The (artificial) example quoted would be represented as:

```
[Nns:s John] expected [Nns:O999 Mary] [Ti:o [s999 GHOST] to admit [Ni:o it]]
```

where the index `999` shows that the ghost acting as logical subject (symbolized `s`) of the "`admit`" clause is coreferential with `Mary` which acts as surface object (`O`) of the "`expected`" clause; the logical object (`o`) of the "`expected`" clause being the infinitival subordinate clause (`Ti`).

In some cases, movement rules displace a constituent into a tagma within which it has no grammatical role (for instance, an adverb which is logically a clause constituent may interrupt the verb group – sequence of auxiliary verbs and main verb – of the clause): in such cases the functiontag is `G` ("guest"). Constituents which do not logically belong below the node which immediately dominates them in surface structure are always given `G` functiontags and indices linking them to their logical position. With that exception (and with one other exception not discussed here relating to co-ordination), functiontagging is used only for immediate constituents of clauses.

B.3 The formtags

The SUSANNE formtags are as follows:

Rootrank formtags

`O`	paragraph
`Oh`	heading
`Ot`	title (e.g. of book)
`Q`	quotation
`I`	interpolation
`Iq`	tag question
`Iu`	technical reference

Clausetags

`S`	main clause
`Ss`	embedded quoting clause
`Fa`	adverbial clause
`Fn`	nominal clause
`Fr`	relative clause
`Ff`	fused relative
`Fc`	comparative clause
`Tg`	present participle clause
`Tn`	past participle clause
`Ti`	infinitival clause
`Tf`	"`for-to`" clause
`Tb`	bare nonfinite clause
`Tq`	infinitival relative clause

W	"with" clause
A	special "as" clause
Z	reduced ("whiz-deleted") relative
L	miscellaneous verbless clause

Phrasetags

V	verb group
N	noun phrase
J	adjective phrase
R	adverb phrase
P	prepositional phrase
D	determiner phrase
M	numeral phrase
G	genitive phrase

The various phrase categories take lower-case subcategory symbols which can be combined in any meaningful combination (e.g. the verb group must have been noticed would be formtagged Vcfp). The phrase subcategories are:

Vo	operator section of verb group, when separated from remainder of V (e.g. by subject-auxiliary inversion)
Vr	remainder of V from which Vo has been separated
Vm	V beginning with am
Va	V beginning with are
Vs	V beginning with was
Vz	V beginning with other 3rd-singular verb
Vw	V beginning with were
Vj	V beginning with be
Vd	V beginning with past tense
Vi	infinitival V
Vg	V beginning with present participle
Vn	V beginning with past participle
Vc	V beginning with modal
Vk	V containing emphatic DO
Ve	negative V
Vf	perfective V
Vu	progressive V
Vp	passive V
Vb	V ending with BE
Vx	V lacking main verb
Vt	catenative V

Nq	wh- N
Nv	wh...ever N
Ne	I/me as whole or head
Ny	you as whole or head
Ni	it as whole or head
Nj	adjectival head
Nn	proper name
Nu	unit of measurement as head
Na	marked as subject

No	marked as nonsubject
Ns	marked as singular
Np	marked as plural
Jq	wh- J
Jv	wh...ever J
Jx	measured absolute J
Jr	measured comparative J
Jh	heavy (postmodified) J
Rq	wh- R
Rv	wh...ever R
Rx	measured absolute R
Rr	measured comparative R
Rs	adverb conducive to asyndeton
Rw	quasi-nominal adverb
Po	"of" phrase
Pb	"by" phrase
Pq	"wh-" P
Pv	"wh...ever" P
Dq	wh- D
Dv	wh...ever D
Ds	marked as singular
Dp	marked as plural
Ms	M headed by one
Gq	wh- G
Gv	wh...ever G

B.4 Non-alphanumeric formtag suffixes

Formtags may also contain non-alphanumeric symbols, including:

?	interrogative clause * imperative clause
%	subjunctive clause
!	exclamatory clause or other item
"	vocative item

Other non-alphanumeric symbols represent co-ordination structure. Under the SUSANNE scheme, second and subsequent conjuncts in a co-ordination are analysed as subordinate to the first conjunct; thus a co-ordination of the form:

```
chi, psi, and omega
```

(whatever the grammatical rank of the word-sequences chi, psi, etc.) would be assigned a structure of the form:

```
[chi, [psi], [and omega]]
```

The formtag of the entire co-ordination is determined by the properties of the first conjunct (except for singular/plural subcategories in the case of phrase categories to which these apply); the later conjuncts (which will often be transformationally reduced) have nodes of their own whose formtags mark them as *subordinate conjuncts*. The following symbols relate to co-ordination (and apposition) structure:

`+`	subordinate conjunct introduced by conjunction
`-`	subordinate conjunct not introduced by conjunction
`@`	appositional element
`&`	co-ordinate structure acting as first conjunct within a higher co-ordination (marked in certain cases only)

Co-ordination is recognised as occurring between words as well as between higher-rank tagmas. Therefore nonterminal nodes may have formtags consisting of wordtags followed by co-ordination symbols, thus (using `WT` to stand for an arbitrary wordtag):

`WT&`	co-ordination of words
`WT+`	conjunct within wordlevel co-ordination that is introduced by a conjunction
`WT-`	conjunct within wordlevel co-ordination not introduced by a conjunction

A wordlevel co-ordination always takes an ampersand on its formtag; phrase or clause co-ordinations do so only in very restricted circumstances.

Also, certain sequences of orthographic words, in certain uses, are regarded as functioning grammatically as single words (*grammatical idioms*). For instance, `none the less` would normally be treated as a grammatical idiom, equivalent to an adverb (for which the wordtag is `RR`). In such cases, the non-terminal node dominating the sequence has a formtag consisting of an equals sign suffixed to the corresponding wordtag; and the individual words composing the grammatical idiom are not wordtagged in their own right, but receive tags with numerical suffixes reflecting their membership of an idiom. The sequence `none the less` would be formtagged `RR=`, and the words `none`, `the`, and `less` in this context would be wordtagged `RR31` `RR32` `RR33`.

Note that formtags of the forms `WT&`, `WT+`, `WT-` and `WT=`, count as wordrank formtags for the purposes of determining tree structure as discussed above.

B.5 The functiontags

Functiontags divide into *complement* and *adjunct* tags: broadly, a given complement tag can occur at most once in any clause, but a clause may contain multiple adjuncts of the same type.

Complement functiontags

`0`	paragraph
`s`	logical subject
`o`	logical direct object
`i`	indirect object
`u`	prepositional object
`e`	predicate complement of subject
`j`	predicate complement of object
`a`	agent of passive
`S`	surface (and not logical) subject

`O` surface (and not logical) direct object
`G` "guest" having no grammatical role within its tagma

Adjunct functiontags
`p` place
`q` direction
`t` time
`h` manner or degree
`m` modality
`c` contingency
`r` respect
`w` comitative
`k` benefactive
`b` absolute

Other functiontags
`n` participle of phrasal verb
`x` relative clause having higher clause as antecedent
`z` complement of catenative

Appendix C

Coverage of SUSANNE grammar on SUSANNE corpus

The following sections show the concrete data we have obtained for all the tests mentioned in chapter 3, and therefore these also are the concrete numbers we have used to plot all the figures that appear in that chapter.

C.1 Ambiguities with SUSANNE grammar on `Notrace` corpus

The following rows of numbers are the concrete data of the ambiguities of the SUSANNE grammar when it parses all the sentences in the `Notrace` corpus. These are the numbers we have used to plot figure 3.1. Each cell contains four numbers: number of ambiguities, number of sentences with that number of ambiguities, percentage of sentences, and cumulative percentage of sentences.

1	846	19.711	19.711	2	497	11.580	31.291	3	198	4.613	35.904
4	225	5.242	41.146	5	133	3.099	44.245	6	149	3.472	47.717
7	42	0.979	48.695	8	136	3.169	51.864	9	55	1.281	53.145
10	87	2.027	55.172	11	13	0.303	55.475	12	76	1.771	57.246
13	16	0.373	57.619	14	25	0.582	58.201	15	31	0.722	58.924
16	62	1.445	60.368	17	7	0.163	60.531	18	42	0.979	61.510
19	5	0.116	61.626	20	68	1.584	63.211	21	15	0.349	63.560
22	19	0.443	64.003	23	4	0.093	64.096	24	56	1.305	65.401
25	19	0.443	65.843	26	11	0.256	66.100	27	13	0.303	66.403
28	14	0.326	66.729	29	5	0.116	66.845	30	35	0.815	67.661
31	3	0.070	67.731	32	39	0.909	68.639	33	6	0.140	68.779
34	6	0.140	68.919	35	2	0.047	68.966	36	28	0.652	69.618
37	2	0.047	69.664	38	7	0.163	69.828	39	3	0.070	69.897
40	32	0.746	70.643	41	3	0.070	70.713	42	21	0.489	71.202
43	2	0.047	71.249	44	10	0.233	71.482	45	11	0.256	71.738
46	6	0.140	71.878	48	30	0.699	72.577	49	2	0.047	72.623
50	14	0.326	72.950	51	2	0.047	72.996	52	6	0.140	73.136
53	1	0.023	73.159	54	15	0.349	73.509	55	2	0.047	73.555
56	11	0.256	73.812	57	4	0.093	73.905	58	3	0.070	73.975
59	3	0.070	74.045	60	23	0.536	74.581	61	1	0.023	74.604
62	3	0.070	74.674	63	2	0.047	74.720	64	17	0.396	75.116
65	2	0.047	75.163	66	6	0.140	75.303	67	1	0.023	75.326
68	4	0.093	75.419	69	2	0.047	75.466	70	4	0.093	75.559
72	15	0.349	75.909	74	1	0.023	75.932	75	3	0.070	76.002
76	4	0.093	76.095	77	2	0.047	76.142	78	4	0.093	76.235
79	3	0.070	76.305	80	20	0.466	76.771	82	2	0.047	76.817
83	1	0.023	76.841	84	11	0.256	77.097	85	1	0.023	77.120
86	1	0.023	77.144	87	1	0.023	77.167	88	7	0.163	77.330
89	2	0.047	77.377	90	17	0.396	77.773	91	1	0.023	77.796
92	1	0.023	77.819	93	1	0.023	77.842	94	3	0.070	77.912
96	16	0.373	78.285	99	3	0.070	78.355	100	12	0.280	78.635
101	1	0.023	78.658	102	8	0.186	78.844	104	5	0.116	78.961
105	6	0.140	79.101	106	1	0.023	79.124	107	1	0.023	79.147
108	3	0.070	79.217	110	4	0.093	79.310	111	1	0.023	79.334
112	6	0.140	79.473	114	1	0.023	79.497	116	2	0.047	79.543
118	1	0.023	79.567	120	21	0.489	80.056	121	1	0.023	80.079
123	1	0.023	80.103	124	1	0.023	80.126	126	3	0.070	80.196
128	4	0.093	80.289	129	1	0.023	80.312	130	3	0.070	80.382
132	1	0.023	80.405	134	1	0.023	80.429	135	7	0.163	80.592
136	2	0.047	80.638	140	2	0.047	80.685	141	2	0.047	80.732
142	2	0.047	80.778	144	12	0.280	81.058	146	1	0.023	81.081
147	1	0.023	81.104	148	2	0.047	81.151	149	1	0.023	81.174
150	11	0.256	81.431	152	2	0.047	81.477	154	1	0.023	81.500
156	7	0.163	81.664	158	1	0.023	81.687	159	1	0.023	81.710
160	8	0.186	81.897	162	4	0.093	81.990	163	1	0.023	82.013
164	3	0.070	82.083	165	2	0.047	82.130	166	1	0.023	82.153
168	6	0.140	82.293	170	1	0.023	82.316	171	1	0.023	82.339
175	1	0.023	82.363	176	5	0.116	82.479	177	1	0.023	82.502
180	8	0.186	82.689	181	1	0.023	82.712	182	1	0.023	82.735
184	2	0.047	82.782								

186	1	0.023	82.805	187	1	0.023	82.829	188	1	0.023	82.852	189	3	0.070	82.922
191	1	0.023	82.945	192	7	0.163	83.108	194	1	0.023	83.131	196	2	0.047	83.178
197	1	0.023	83.201	198	3	0.070	83.271	200	8	0.186	83.458	201	1	0.023	83.481
204	3	0.070	83.551	208	3	0.070	83.621	210	6	0.140	83.760	214	1	0.023	83.784
215	1	0.023	83.807	216	3	0.070	83.877	220	2	0.047	83.924	222	1	0.023	83.947
224	3	0.070	84.017	225	4	0.093	84.110	227	1	0.023	84.133	230	3	0.070	84.203
231	1	0.023	84.226	232	1	0.023	84.250	233	1	0.023	84.273	234	1	0.023	84.296
235	1	0.023	84.320	240	13	0.303	84.623	242	1	0.023	84.646	245	1	0.023	84.669
248	2	0.047	84.716	250	1	0.023	84.739	252	3	0.070	84.809	253	1	0.023	84.832
256	4	0.093	84.925	258	1	0.023	84.949	260	2	0.047	84.995	264	2	0.047	85.042
270	5	0.116	85.158	272	2	0.047	85.205	274	1	0.023	85.228	275	1	0.023	85.252
280	8	0.186	85.438	281	1	0.023	85.461	285	2	0.047	85.508	288	2	0.047	85.555
294	1	0.023	85.578	300	5	0.116	85.694	305	1	0.023	85.718	308	1	0.023	85.741
312	1	0.023	85.764	315	2	0.047	85.811	316	1	0.023	85.834	320	8	0.186	86.021
322	1	0.023	86.044	324	1	0.023	86.067	326	1	0.023	86.090	330	3	0.070	86.160
336	2	0.047	86.207	340	2	0.047	86.253	344	2	0.047	86.300	346	1	0.023	86.323
348	1	0.023	86.347	349	1	0.023	86.370	350	1	0.023	86.393	351	1	0.023	86.417
354	1	0.023	86.440	357	1	0.023	86.463	358	1	0.023	86.486	360	5	0.116	86.603
364	1	0.023	86.626	368	1	0.023	86.650	372	1	0.023	86.673	375	1	0.023	86.696
378	1	0.023	86.719	380	1	0.023	86.743	382	1	0.023	86.766	384	3	0.070	86.836
385	1	0.023	86.859	392	3	0.070	86.929	393	1	0.023	86.952	396	2	0.047	86.999
400	2	0.047	87.046	412	1	0.023	87.069	414	1	0.023	87.092	416	2	0.047	87.139
420	1	0.023	87.162	430	2	0.047	87.209	432	4	0.093	87.302	436	1	0.023	87.325
440	1	0.023	87.349	448	2	0.047	87.395	450	1	0.023	87.418	455	1	0.023	87.442
476	1	0.023	87.465	480	10	0.233	87.698	489	1	0.023	87.721	490	2	0.047	87.768
496	1	0.023	87.791	500	6	0.140	87.931	501	1	0.023	87.954	504	1	0.023	87.978
510	1	0.023	88.001	512	1	0.023	88.024	514	1	0.023	88.048	516	2	0.047	88.094
519	1	0.023	88.117	521	1	0.023	88.141	525	1	0.023	88.164	528	3	0.070	88.234
530	1	0.023	88.257	538	1	0.023	88.281	539	1	0.023	88.304	540	2	0.047	88.350
544	1	0.023	88.374	546	1	0.023	88.397	548	1	0.023	88.420	550	1	0.023	88.444
560	2	0.047	88.490	567	1	0.023	88.514	572	1	0.023	88.537	575	1	0.023	88.560
576	5	0.116	88.677	580	1	0.023	88.700	586	1	0.023	88.723	590	1	0.023	88.747
594	1	0.023	88.770	600	4	0.093	88.863	615	1	0.023	88.886	630	2	0.047	88.933
640	1	0.023	88.956	644	1	0.023	88.979	645	1	0.023	89.003	648	1	0.023	89.026
650	1	0.023	89.049	652	1	0.023	89.073	654	1	0.023	89.096	660	1	0.023	89.119
666	1	0.023	89.143	672	1	0.023	89.166	680	1	0.023	89.189	683	1	0.023	89.212
690	1	0.023	89.236	693	1	0.023	89.259	700	2	0.047	89.306	702	3	0.070	89.376
708	1	0.023	89.399	712	2	0.047	89.445	716	1	0.023	89.469	720	3	0.070	89.539
722	1	0.023	89.562	748	1	0.023	89.585	756	3	0.070	89.655	765	1	0.023	89.678
768	5	0.116	89.795	780	1	0.023	89.818	800	4	0.093	89.911	810	1	0.023	89.935
812	1	0.023	89.958	832	3	0.070	90.028	835	1	0.023	90.051	840	2	0.047	90.098
842	1	0.023	90.121	864	3	0.070	90.191	882	1	0.023	90.214	892	1	0.023	90.238
900	2	0.047	90.284	930	1	0.023	90.308	936	2	0.047	90.354	945	1	0.023	90.377
960	4	0.093	90.471	968	1	0.023	90.494	972	1	0.023	90.517	978	1	0.023	90.541
981	1	0.023	90.564	998	1	0.023	90.587	1000	2	0.047	90.634	1008	1	0.023	90.657
1024	1	0.023	90.680	1032	1	0.023	90.704	1050	1	0.023	90.727	1056	1	0.023	90.750
1064	1	0.023	90.774	1080	3	0.070	90.843	1092	1	0.023	90.867	1104	1	0.023	90.890
1111	1	0.023	90.913	1112	1	0.023	90.937	1120	2	0.047	90.983	1122	1	0.023	91.007
1128	1	0.023	91.030	1134	1	0.023	91.053	1144	1	0.023	91.076	1148	1	0.023	91.100
1152	2	0.047	91.146	1155	1	0.023	91.170	1170	1	0.023	91.193	1176	1	0.023	91.216
1184	1	0.023	91.240	1200	3	0.070	91.309	1244	1	0.023	91.333	1248	3	0.070	91.403
1260	3	0.070	91.473	1272	1	0.023	91.496	1280	2	0.047	91.542	1296	1	0.023	91.566
1300	1	0.023	91.589	1315	1	0.023	91.612	1320	1	0.023	91.636	1340	1	0.023	91.659
1350	1	0.023	91.682	1352	1	0.023	91.705	1355	1	0.023	91.729	1360	2	0.047	91.775
1368	1	0.023	91.799	1400	2	0.047	91.845	1404	2	0.047	91.892	1424	1	0.023	91.915
1440	4	0.093	92.008	1452	1	0.023	92.032	1470	1	0.023	92.055	1504	1	0.023	92.078
1512	1	0.023	92.102	1538	1	0.023	92.125	1550	1	0.023	92.148	1572	1	0.023	92.171
1584	2	0.047	92.218	1592	1	0.023	92.241	1596	1	0.023	92.265	1626	1	0.023	92.288
1632	1	0.023	92.311	1640	1	0.023	92.335	1680	2	0.047	92.381	1712	1	0.023	92.404
1736	1	0.023	92.428	1752	1	0.023	92.451	1800	2	0.047	92.498	1820	1	0.023	92.521
1848	1	0.023	92.544	1860	1	0.023	92.568	1872	1	0.023	92.591	1917	1	0.023	92.614
1920	2	0.047	92.661	1936	1	0.023	92.684	1944	1	0.023	92.707	1952	1	0.023	92.731
1972	1	0.023	92.754	1974	1	0.023	92.777	1980	1	0.023	92.801	2000	1	0.023	92.824
2016	2	0.047	92.870	2024	1	0.023	92.894	2025	1	0.023	92.917	2052	1	0.023	92.940
2088	1	0.023	92.964	2096	1	0.023	92.987	2128	1	0.023	93.010	2140	1	0.023	93.034
2160	2	0.047	93.080	2166	1	0.023	93.103	2176	2	0.047	93.150	2178	1	0.023	93.173
2190	1	0.023	93.197	2218	1	0.023	93.220	2250	1	0.023	93.243	2259	1	0.023	93.267
2274	1	0.023	93.290	2288	1	0.023	93.313	2320	2	0.047	93.360	2336	1	0.023	93.383
2340	1	0.023	93.406	2361	1	0.023	93.430	2400	3	0.070	93.500	2435	1	0.023	93.523
2440	1	0.023	93.546	2448	2	0.047	93.593	2480	1	0.023	93.616	2532	1	0.023	93.639
2560	1	0.023	93.663	2600	1	0.023	93.686	2640	1	0.023	93.709	2658	1	0.023	93.733
2680	1	0.023	93.756	2688	1	0.023	93.779	2704	1	0.023	93.802	2720	1	0.023	93.826
2756	1	0.023	93.849	2772	1	0.023	93.872	2854	1	0.023	93.896	2880	3	0.070	93.966

2916	1	0.023	93.989	2952	1	0.023	94.012	2970	2	0.047	94.059	3072	1	0.023	94.082
3076	1	0.023	94.105	3140	1	0.023	94.129	3150	1	0.023	94.152	3208	1	0.023	94.175
3234	1	0.023	94.199	3244	1	0.023	94.222	3264	1	0.023	94.245	3312	1	0.023	94.268
3360	1	0.023	94.292	3388	1	0.023	94.315	3456	1	0.023	94.338	3468	1	0.023	94.362
3472	1	0.023	94.385	3586	1	0.023	94.408	3630	1	0.023	94.432	3639	1	0.023	94.455
3756	1	0.023	94.478	3958	1	0.023	94.501	3960	1	0.023	94.525	3994	1	0.023	94.548
3996	1	0.023	94.571	4007	1	0.023	94.595	4032	1	0.023	94.618	4080	1	0.023	94.641
4087	1	0.023	94.664	4160	1	0.023	94.688	4320	1	0.023	94.711	4366	1	0.023	94.734
4400	1	0.023	94.758	4480	1	0.023	94.781	4485	1	0.023	94.804	4536	2	0.047	94.851
4600	1	0.023	94.874	4608	1	0.023	94.897	4644	1	0.023	94.921	4680	1	0.023	94.944
4704	1	0.023	94.967	4784	1	0.023	94.991	4800	2	0.047	95.037	4809	1	0.023	95.061
4815	1	0.023	95.084	4816	1	0.023	95.107	4832	1	0.023	95.130	4876	1	0.023	95.154
5040	1	0.023	95.177	5130	1	0.023	95.200	5280	1	0.023	95.224	5338	1	0.023	95.247
5568	1	0.023	95.270	5616	1	0.023	95.294	5670	1	0.023	95.317	5760	2	0.047	95.363
5796	1	0.023	95.387	6048	2	0.047	95.433	6336	1	0.023	95.457	6656	1	0.023	95.480
6672	1	0.023	95.503	6732	1	0.023	95.527	6750	1	0.023	95.550	6840	2	0.047	95.596
6912	1	0.023	95.620	7080	1	0.023	95.643	7096	1	0.023	95.666	7200	1	0.023	95.690
7260	1	0.023	95.713	7480	1	0.023	95.736	7560	2	0.047	95.783	7692	1	0.023	95.806
7696	1	0.023	95.829	7872	1	0.023	95.853	7875	1	0.023	95.876	7884	1	0.023	95.899
8064	1	0.023	95.923	8316	1	0.023	95.946	8580	1	0.023	95.969	8624	1	0.023	95.993
8643	1	0.023	96.016	8802	1	0.023	96.039	8925	1	0.023	96.062	8988	1	0.023	96.086
9000	1	0.023	96.109	9088	1	0.023	96.132	9360	1	0.023	96.156	9400	1	0.023	96.179
9520	1	0.023	96.202	9600	1	0.023	96.226	10100	1	0.023	96.249	10512	1	0.023	96.272
11696	1	0.023	96.295	12096	1	0.023	96.319	12600	2	0.047	96.365	12640	1	0.023	96.389
13048	1	0.023	96.412	13500	1	0.023	96.435	13598	1	0.023	96.459	13776	1	0.023	96.482
13824	1	0.023	96.505	14016	1	0.023	96.528	14112	1	0.023	96.552	14208	1	0.023	96.575
14480	1	0.023	96.598	14652	1	0.023	96.622	14751	1	0.023	96.645	14760	1	0.023	96.668
15120	1	0.023	96.692	15166	1	0.023	96.715	15360	1	0.023	96.738	15680	1	0.023	96.761
15960	1	0.023	96.785	16640	1	0.023	96.808	16800	1	0.023	96.831	17280	1	0.023	96.855
17490	1	0.023	96.878	17496	1	0.023	96.901	17640	1	0.023	96.925	17784	1	0.023	96.948
17850	1	0.023	96.971	17920	1	0.023	96.994	18000	1	0.023	97.018	19350	1	0.023	97.041
20560	1	0.023	97.064	20736	1	0.023	97.088	20964	1	0.023	97.111	21600	1	0.023	97.134
21808	1	0.023	97.158	21848	1	0.023	97.181	21950	1	0.023	97.204	22451	1	0.023	97.227
22488	1	0.023	97.251	22680	1	0.023	97.274	23816	1	0.023	97.297	25984	1	0.023	97.321
26218	1	0.023	97.344	26964	1	0.023	97.367	31542	1	0.023	97.390	31696	1	0.023	97.414
32080	1	0.023	97.437	35600	1	0.023	97.460	36000	1	0.023	97.484	36408	1	0.023	97.507
37440	1	0.023	97.530	37586	1	0.023	97.554	38112	1	0.023	97.577	39420	1	0.023	97.600
40950	1	0.023	97.623	41040	1	0.023	97.647	51450	1	0.023	97.670	53040	1	0.023	97.693
53100	1	0.023	97.717	54728	1	0.023	97.740	54800	1	0.023	97.763	56430	1	0.023	97.787
56704	1	0.023	97.810	58425	1	0.023	97.833	63288	1	0.023	97.856	63404	1	0.023	97.880
63468	1	0.023	97.903	65064	1	0.023	97.926	68880	1	0.023	97.950	71265	1	0.023	97.973
72576	1	0.023	97.996	74470	1	0.023	98.020	74520	1	0.023	98.043	76916	1	0.023	98.066
82800	1	0.023	98.089	86240	1	0.023	98.113	87175	1	0.023	98.136	87192	1	0.023	98.159
87360	1	0.023	98.183	88226	1	0.023	98.206	88704	1	0.023	98.229	101304	1	0.023	98.253
116640	1	0.023	98.276	118800	1	0.023	98.299	120428	1	0.023	98.322	123202	1	0.023	98.346
126345	1	0.023	98.369	136080	1	0.023	98.392	152568	1	0.023	98.416	153090	1	0.023	98.439
158340	1	0.023	98.462	163800	1	0.023	98.486	166360	1	0.023	98.509	168180	1	0.023	98.532
183804	1	0.023	98.555	184992	1	0.023	98.579	200000	1	0.023	98.602	205440	1	0.023	98.625
224000	1	0.023	98.649	225000	1	0.023	98.672	226842	1	0.023	98.695	226848	1	0.023	98.719
228096	1	0.023	98.742	243648	1	0.023	98.765	253368	1	0.023	98.788	262848	1	0.023	98.812
267264	1	0.023	98.835	285870	1	0.023	98.858	307704	1	0.023	98.882	310080	1	0.023	98.905
314496	1	0.023	98.928	340200	1	0.023	98.952	372240	1	0.023	98.975	392156	1	0.023	98.998
437904	1	0.023	99.021	471744	1	0.023	99.045	485184	1	0.023	99.068	595350	1	0.023	99.091
618240	1	0.023	99.115	656502	1	0.023	99.138	792480	1	0.023	99.161	793800	1	0.023	99.185
800208	1	0.023	99.208	870912	1	0.023	99.231	882792	1	0.023	99.254	1003520	1	0.023	99.278
1332918	1	0.023	99.301	1394800	1	0.023	99.324	1436400	1	0.023	99.348	1843200	1	0.023	99.371
1940400	1	0.023	99.394	1946112	1	0.023	99.418	1952588	1	0.023	99.441	2070474	1	0.023	99.464
2552760	1	0.023	99.487	2565600	1	0.023	99.511	2569728	1	0.023	99.534	2711376	1	0.023	99.557
3464208	1	0.023	99.581	3892328	1	0.023	99.604	3995708	1	0.023	99.627	15372816	1	0.023	99.651
16907403	1	0.023	99.674	27447552	1	0.023	99.697	36834000	1	0.023	99.720	38707200	1	0.023	99.744
39053720	1	0.023	99.767	50101216	1	0.023	99.790	66772134	1	0.023	99.814	419126400	1	0.023	99.837
1792391040	1	0.023	99.860	2190154400	1	0.023	99.884	overflow	5	0.116	100.000				

Cell `[overflow 5 0.116 100.000]` means that there are 5 sentences for which the number of ambiguities is greater than $2^{32} = 4.294967\text{e+}09$, the greatest value of an `unsigned long int` in our system.

The average of ambiguities per sentence, without considering those 5 sentences with *overflow*, is:

$$\#\,average\,of\,ambiguities = \frac{\#\,ambiguities}{\#\,sentences} = \frac{4748032872}{4292-5} \approx 1107542\,ambiguities\,per\,sentence$$

However, it is much more interesting to calculate the median, that is, the point which leaves half of the population on its left and the other half on its right. Last numbers in cells `[7 42 0.979 48.695]` and `[8 136 3.169 51.864]` yield a median of something between 7 and 8 ambiguities per sentence.

C.2 Succesful with SUSANNE grammar on `Notrace` corpus

The following rows of numbers are the concrete data of the succesful of the SUSANNE grammar when it parses all the sentences in the `Notrace` corpus. These are the numbers we have used to plot figure 3.2. Each cell contains four numbers: rank of the reference tree among all the trees provided by the parser, number of sentences with that rank, percentage of sentences, and cumulative percentage of sentences.

1	3335	77.703	77.703	2	447	10.415	88.117	3	118	2.749	90.867	4	59	1.375	92.241
5	17	0.396	92.637	6	22	0.513	93.150	7	20	0.466	93.616	8	11	0.256	93.872
9	11	0.256	94.129	10	7	0.163	94.292	11	3	0.070	94.362	12	5	0.116	94.478
13	4	0.093	94.571	16	1	0.023	94.595	17	3	0.070	94.664	18	1	0.023	94.688
19	1	0.023	94.711	20	2	0.047	94.758	21	1	0.023	94.781	22	1	0.023	94.804
23	1	0.023	94.828	25	3	0.070	94.897	27	3	0.070	94.967	28	1	0.023	94.991
29	1	0.023	95.014	31	1	0.023	95.037	38	1	0.023	95.061	61	1	0.023	95.084
67	1	0.023	95.107	79	1	0.023	95.130	322	1	0.023	95.154	nf	208	4.846	100.000

Cell `[nf 208 4.846 100.000]` means that there are 208 sentences for which the number of ambiguities is greater than 5,000 (an artificial limit set by us in order to reduce computation time), and therefore the rank of the reference tree among all the parsings provided by the parser is `not found`.

The average of the rank, without considering those 208 *not found* sentences, is:

$$\#\, average\, of\, rank = \frac{sum\, of\, rankings}{\#\, sentences} = \frac{6599}{4292 - 208} = 1.61$$

but the last number in the cell `[1 3335 77.703 77.703]` shows that the parsing with the highest probability is equal to the reference tree for 77.703% of the sentences.

C.3 Ambiguities with SUSANNE grammar on `Trace` corpus

The following rows of numbers are the concrete data of the ambiguities of the SUSANNE grammar when it parses all the sentences in the `Trace` corpus. Each cell contains four numbers: number of ambiguities, number of sentences with that number of ambiguities, percentage of sentences, and cumulative percentage of sentences.

np	2128	97.258	97.258	1	6	0.274	97.532	2	9	0.411	97.943	4	3	0.137	98.080
6	2	0.091	98.172	7	1	0.046	98.218	8	1	0.046	98.263	12	2	0.091	98.355
24	3	0.137	98.492	26	1	0.046	98.537	30	1	0.046	98.583	37	1	0.046	98.629
46	1	0.046	98.675	103	1	0.046	98.720	110	1	0.046	98.766	112	1	0.046	98.812
136	1	0.046	98.857	145	1	0.046	98.903	187	1	0.046	98.949	245	1	0.046	98.995
334	1	0.046	99.040	396	1	0.046	99.086	441	1	0.046	99.132	714	1	0.046	99.177
720	1	0.046	99.223	864	1	0.046	99.269	1238	1	0.046	99.314	1441	1	0.046	99.360
2448	1	0.046	99.406	2960	1	0.046	99.452	3885	1	0.046	99.497	6034	1	0.046	99.543
6163	1	0.046	99.589	11291	1	0.046	99.634	21754	1	0.046	99.680	25962	1	0.046	99.726
71500	1	0.046	99.771	76920	1	0.046	99.817	288192	1	0.046	99.863	504630	1	0.046	99.909
2291058	1	0.046	99.954	overflow	1	0.046	100.000								

Cell `[np 2128 97.258 97.258]` means that there are 2,128 `not parsed` sentences, i.e. the SUSANNE grammar is able to parse only 60 sentences of the `Trace` corpus.

Cell `[overflow 1 0.046 100.000]` means that there is 1 sentence for which the number of ambiguities is greater than $2^{32} = 4.294967e{+}09$, the greatest value of an `unsigned long int` in our system.

The average number of ambiguities per sentence, without considering those sentences with *overflow* or *not parsed*, is:

$$\#\ average\ of\ ambiguities = \frac{\#\ ambiguities}{\#\ sentences} = \frac{3320281}{59} \approx 56276\ ambiguities\ per\ sentence$$

If we add the number of sentences with 30 ambiguities or less, we obtain 6+9+3+2+1+1+2+3+1+1 = 29 sentences, which is less than half of the parseable sentences (29.5). And if we add the number of sentences with 37 ambiguities, the next value for the number of ambiguities in the distribution, we obtain 29 + 1 = 30 sentences, that is, more than half of the population. This yields a median of something between 30 and 37 ambiguities per sentence.

C.4 Succes with SUSANNE grammar on Trace corpus

Only 60 sentences of the TRACE corpus can be parsed with the SUSANNE grammar, and of course none of these has a tree matching the tree in the reference corpus. This is because all the trees provided by the parser do not have traces, and all the trees present in the reference corpus for these sentences do have traces.

C.5 Cross products on Trace corpus

The following rows of numbers are the concrete data of the cross products of the number of tags per words for each sentence in the TRACE corpus. These are the numbers we have used to plot figure 4.1. Each cell contains five numbers: number of ambiguities, number of sentences with that number of ambiguities, average of words per sentence, percentage of sentences, and cumulative percentage of sentences.

1e+0	1	6	0.046	0.046	1e+1	22	7	1.005	1.051	1e+2	80	9	3.656	4.707	1e+3	150	12	6.856	11.563
1e+4	181	15	8.272	19.835	1e+5	240	19	10.969	30.804	1e+6	218	21	9.963	40.768	1e+7	214	25	9.781	50.548
1e+8	211	28	9.644	60.192	1e+9	180	31	8.227	68.419	1e+10	173	32	7.907	76.325	1e+11	124	35	5.667	81.993
1e+12	84	38	3.839	85.832	1e+13	76	43	3.473	89.305	1e+14	59	44	2.697	92.002	1e+15	41	47	1.874	93.876
1e+16	42	51	1.920	95.795	1e+17	12	54	0.548	96.344	1e+18	16	59	0.731	97.075	1e+19	7	56	0.320	97.395
1e+20	17	61	0.777	98.172	1e+21	10	65	0.457	98.629	1e+22	5	69	0.229	98.857	1e+23	3	65	0.137	98.995
1e+24	4	73	0.183	99.177	1e+25	5	80	0.229	99.406	1e+26	3	84	0.137	99.543	1e+27	1	82	0.046	99.589
1e+28	2	113	0.091	99.680	1e+30	1	132	0.046	99.726	1e+33	2	128	0.091	99.817	1e+34	1	138	0.046	99.863
1e+36	1	116	0.046	99.909	1e+42	1	166	0.046	99.954	1e+44	1	154	0.046	100.000					

The average number of words per sentence is 27, and the median is 25. The average number of ambiguities per sentence is 1.50e+40, and the median is 1e+6.

Appendix D

Forest for 8 examples of non-parseable traced sentences

The following lines are the forests of sequences of tags for the 8 examples of sentences cited in chapter 3. There are two kinds of lines. The last line contains the mark `Ref` and the list of tags that appear in the reference corpus. The rest of the lines have the following format: line number, length of the sequence, starting point of the sequence (i.e. position of the first word covered by the sequence inside the sentence), number of subtrees covering the same piece of sentence with the same sequence of tags, probability of the best subtree, and the list of tags.

Forest and reference for `trace_np_sample_1` (the 4-word sentence in figures 3.3 and 3.4):

```
943 1 1 13 (-5.011376) :389
943 1 2 7 (-0.323840) :266
943 1 3 24 (-8.976890) :164
943 1 3 13 (-6.803134) :389
943 1 4 8 (-5.612335) :106
943 1 4 5 (-6.196444) :182
943 1 4 7 (-2.100265) :332
943 1 4 1 (-5.159055) :336
943 2 1 3 (-7.763267) :389 :266
943 2 3 23 (-9.930394) :389 :332
943 2 3 10 (-20.481749) :389 :182
943 2 3 8 (-21.424752) :389 :106
943 2 3 4 (-29.257421) :164 :182
943 2 3 9 (-24.463031) :164 :106
943 2 3 11 (-20.492888) :164 :332
Ref :389 :266 :389 :332
```

Forest and reference for `trace_np_sample_2` (the 5-word sentence in figures 3.5 and 3.6):

```
30 1 1 1 (-5.190175) :29
30 1 1 1 (-3.610919) :303
30 1 1 4 (-1.501979) :53
30 1 2 1 (-0.098595) :100
30 1 2 5 (-1.907069) :274
30 1 2 1 (-5.265277) :27
30 1 2 3 (-2.397167) :373
30 1 2 2 (-4.110876) :378
30 1 2 3 (-3.601869) :383
30 1 3 13 (-4.451437) :334
30 1 3 1 (-3.213145) :335
30 1 3 1 (-2.674149) :336
30 1 3 5 (-0.103612) :51
30 1 4 1 (-0.924948) :156
30 1 4 1 (-1.386294) :234
30 1 4 3 (0.000000) :275
30 1 4 1 (-4.234103) :97
30 1 5 13 (-5.336798) :389
30 2 2 2 (-16.255127) :27 :51
30 2 2 5 (-13.213882) :27 :334
30 2 4 14 (-7.768233) :275 :389
```

```
30 3 1 11 (-15.730912) :53 :373 :51
Ref :53 :373 :51 :275 :389
```

Forest and reference for `trace_np_sample_3` (the 8-word sentence in figures 3.7 and 3.8):

```
26 1 1 6 (-2.073467) :59
26 1 2 1 (-1.531477) :304
26 1 2 3 (-0.114786) :373
26 1 3 12 (-7.932960) :133
26 1 3 13 (-3.616116) :334
26 1 4 13 (-7.370231) :399
26 1 5 1 (-3.215232) :107
26 1 5 1 (-2.477634) :108
26 1 5 2 (-0.027588) :116
26 1 5 1 (0.000000) :15
26 1 5 1 (0.000000) :17
26 1 5 1 (-2.639057) :23
26 1 5 1 (-0.693147) :24
26 1 5 1 (-0.693147) :26
26 1 5 1 (-0.141314) :27
26 1 5 1 (0.000000) :30
26 1 5 1 (-0.081917) :315
26 1 5 1 (-2.856470) :335
26 1 5 1 (-0.057159) :36
26 1 5 3 (-7.225482) :373
26 1 6 1 (-4.204691) :425
26 1 6 1 (0.000000) :54
26 1 6 1 (-0.054068) :64
26 1 6 1 (-0.083382) :68
26 1 6 1 (-0.405465) :70
26 1 6 1 (-0.133531) :86
26 1 6 1 (-3.881562) :91
26 1 6 5 (-4.327440) :96
26 1 6 1 (-0.205020) :9
26 1 7 12 (-5.099747) :133
26 1 8 24 (-6.779670) :164
26 2 3 2 (-21.408569) :133 :399
26 2 3 12 (-13.364653) :334 :399
26 2 5 1 (-8.631693) :116 :96
26 2 6 1 (-5.997914) :9 :133
26 2 7 9 (-13.265711) :133 :164
26 3 1 22 (-21.303223) :59 :373 :133
26 3 6 11 (-14.649386) :9 :133 :164
26 4 1 22 (-43.932255) :59 :373 :133 :399
26 4 5 12 (-17.450961) :116 :9 :133 :164
26 4 5 7 (-20.083995) :27 :9 :133 :164
26 5 4 11 (-40.233577) :399 :373 :9 :133 :164
26 5 4 4 (-32.251916) :399 :116 :9 :133 :164
26 6 3 22 (-45.864596) :334 :399 :373 :9 :133 :164
Ref :59 :373 :334 :399 :116 :9 :133 :164
```

Forest and reference for `trace_np_sample_4` (the 9-word sentence in figures 3.9 and 3.10):

```
16 1 1 1 (0.000000) :126
16 1 1 13 (-8.034955) :334
16 1 1 1 (-4.465904) :335
16 1 1 5 (-2.359280) :51
16 1 2 3 (0.000000) :264
16 1 2 2 (-0.098246) :6
16 1 3 8 (-0.678619) :215
16 1 4 13 (-5.336798) :389
16 1 5 1 (-0.038578) :105
16 1 5 1 (-3.310542) :108
16 1 5 1 (-0.475423) :303
16 1 5 1 (-5.159055) :335
16 1 5 1 (-0.530629) :33
16 1 6 1 (-1.531477) :304
16 1 6 3 (-0.114786) :373
16 1 7 1 (-0.117230) :11
16 1 7 1 (-0.635988) :252
16 1 7 1 (-5.159055) :335
16 1 7 6 (-2.890371) :363
16 1 7 1 (0.000000) :79
```

```
16 1 8 1 (-3.772761) :335
16 1 8 1 (-3.549619) :336
16 1 8 5 (-0.184228) :44
16 1 9 22 (-5.147989) :167
16 2 2 1 (-2.163159) :6 :215
16 2 5 2 (-2.006900) :303 :304
16 2 7 11 (-8.772525) :335 :336
16 2 7 5 (-0.994605) :11 :44
16 3 7 6 (-15.169133) :11 :44 :167
16 3 7 19 (-20.658754) :335 :336 :167
16 4 5 10 (-21.985757) :303 :304 :335 :336
Ref :51 :6 :215 :389 :105 :373 :11 :44 :167
```

Forest and reference for `trace_np_sample_5` (the 9-word sentence in figures 3.11 and 3.12):

```
32 1 1 1 (-5.190175) :29
32 1 1 1 (-3.610919) :303
32 1 1 4 (-1.501979) :53
32 1 2 1 (-0.098595) :100
32 1 2 5 (-1.907069) :274
32 1 2 1 (-5.265277) :27
32 1 2 3 (-2.397167) :373
32 1 2 2 (-4.110876) :378
32 1 2 3 (-3.601869) :383
32 1 3 1 (0.000000) :123
32 1 3 5 (-0.256353) :75
32 1 3 1 (0.000000) :81
32 1 4 3 (-0.332079) :272
32 1 5 3 (-2.277541) :386
32 1 6 12 (-3.847242) :388
32 1 7 6 (-0.275788) :287
32 1 7 1 (0.000000) :365
32 1 8 1 (-1.661885) :108
32 1 8 1 (-2.186047) :111
32 1 8 1 (-0.011651) :119
32 1 8 1 (-0.154151) :320
32 1 8 8 (-5.934895) :325
32 1 8 1 (0.000000) :342
32 1 8 3 (-0.015164) :362
32 1 9 14 (-0.074503) :374
32 2 5 1 (-7.984980) :386 :388
32 2 6 13 (-5.605905) :388 :287
32 2 8 6 (-3.943355) :362 :374
32 3 3 1 (-5.987649) :75 :272 :386
32 3 4 11 (-17.141129) :272 :386 :388
32 4 3 1 (-11.695088) :75 :272 :386 :388
Ref :53 :373 :75 :272 :386 :388 :287 :362 :374
```

Forest and reference for `trace_np_sample_6` (the 12-word sentence in figures 3.13 and 3.14):

```
61 1 1 5 (-3.839451) :329
61 1 1 1 (-0.958137) :89
61 1 2 1 (-1.531477) :304
61 1 2 3 (-0.114786) :373
61 1 3 1 (-4.204691) :425
61 1 3 1 (0.000000) :54
61 1 3 1 (-0.054068) :64
61 1 3 1 (-0.083382) :68
61 1 3 1 (-0.405465) :70
61 1 3 1 (-0.133531) :86
61 1 3 1 (-3.881562) :91
61 1 3 5 (-4.327440) :96
61 1 3 1 (-0.205020) :9
61 1 4 12 (-9.031575) :133
61 1 5 22 (-5.907094) :167
61 1 5 18 (-6.154859) :396
61 1 6 8 (-0.957545) :106
61 1 6 1 (-5.613127) :107
61 1 6 1 (-0.240141) :109
61 1 6 1 (-0.105361) :12
61 1 6 1 (-1.722769) :22
61 1 6 1 (-0.693147) :24
61 1 6 7 (-2.358674) :332
```

```
61 1 6 1 (-2.068011) :335
61 1 7 1 (0.000000) :113
61 1 7 1 (-1.098613) :338
61 1 7 1 (0.000000) :62
61 1 7 1 (-0.348307) :66
61 1 7 1 (-0.120895) :8
61 1 8 22 (-5.442788) :167
61 1 9 1 (-0.737931) :108
61 1 9 1 (-0.329753) :111
61 1 9 1 (0.000000) :115
61 1 9 1 (-0.005837) :121
61 1 9 1 (-5.295815) :20
61 1 9 1 (0.000000) :323
61 1 9 1 (-1.981002) :335
61 1 9 1 (-3.772761) :336
61 1 10 9 (-6.968850) :168
61 1 11 8 (-4.620694) :106
61 1 12 2 (-1.252764) :171
61 2 3 1 (-9.929742) :9 :133
61 2 4 10 (-17.135894) :133 :167
61 2 4 5 (-23.749121) :133 :396
61 2 5 6 (-13.573910) :396 :332
61 2 5 12 (-14.882070) :396 :106
61 2 5 11 (-16.119316) :167 :332
61 2 5 16 (-18.232368) :167 :106
61 2 7 29 (-5.563683) :8 :167
61 2 9 1 (-14.692413) :20 :168
61 2 9 2 (-7.667834) :121 :168
61 2 10 6 (-24.121701) :168 :106
61 3 3 10 (-17.708638) :9 :133 :167
61 3 4 11 (-26.387653) :133 :167 :332
61 3 4 16 (-30.746994) :133 :396 :332
61 3 4 14 (-32.471554) :133 :396 :106
61 3 4 7 (-28.456628) :133 :167 :106
61 3 6 62 (-10.404568) :106 :8 :167
61 3 8 9 (-20.914845) :167 :121 :168
61 3 8 6 (-21.601539) :167 :20 :168
61 3 9 6 (-24.671102) :121 :168 :106
61 4 3 9 (-27.468963) :9 :133 :167 :106
61 4 3 11 (-27.403604) :9 :133 :167 :332
61 4 5 136 (-19.527818) :396 :106 :8 :167
61 4 5 5 (-20.945460) :396 :332 :8 :167
61 4 5 158 (-21.635421) :167 :106 :8 :167
61 4 7 26 (-18.412574) :8 :167 :121 :168
61 4 7 9 (-26.635089) :8 :167 :20 :168
61 4 8 24 (-32.879581) :167 :121 :168 :106
61 4 8 3 (-38.238776) :167 :20 :168 :106
61 5 1 12 (-27.353584) :89 :373 :9 :133 :167
61 5 4 5 (-40.630887) :133 :396 :332 :8 :167
61 5 4 210 (-35.874608) :133 :396 :106 :8 :167
61 5 4 134 (-29.387019) :133 :167 :106 :8 :167
61 5 6 29 (-31.440119) :106 :8 :167 :20 :168
61 5 6 62 (-21.150808) :106 :8 :167 :121 :168
61 5 7 5 (-46.048126) :8 :167 :20 :168 :106
61 5 7 70 (-32.307327) :8 :167 :121 :168 :106
61 6 1 23 (-35.563897) :89 :373 :9 :133 :167 :106
61 6 3 110 (-30.872016) :9 :133 :167 :106 :8 :167
61 6 5 68 (-41.145137) :396 :106 :8 :167 :20 :168
61 6 5 256 (-30.274057) :396 :106 :8 :167 :121 :168
61 6 5 10 (-31.511255) :396 :332 :8 :167 :121 :168
61 6 5 82 (-42.911310) :167 :106 :8 :167 :20 :168
61 6 5 158 (-32.381661) :167 :106 :8 :167 :121 :168
61 6 6 14 (-50.870705) :106 :8 :167 :20 :168 :106
61 6 6 158 (-34.892568) :106 :8 :167 :121 :168 :106
61 7 2 6 (-41.914696) :373 :9 :133 :167 :106 :8 :167
61 7 4 5 (-51.196682) :133 :396 :332 :8 :167 :121 :168
61 7 4 108 (-57.150496) :133 :396 :106 :8 :167 :20 :168
61 7 4 210 (-46.620847) :133 :396 :106 :8 :167 :121 :168
61 7 4 70 (-50.238432) :133 :167 :106 :8 :167 :20 :168
61 7 4 134 (-40.133258) :133 :167 :106 :8 :167 :121 :168
61 7 5 25 (-44.596469) :396 :332 :8 :167 :121 :168 :106
61 7 5 44 (-54.935008) :396 :106 :8 :167 :20 :168 :106
61 7 5 600 (-43.226340) :396 :106 :8 :167 :121 :168 :106
```

```
61 7 5 50 (-57.071284) :167 :106 :8 :167 :20 :168 :106
61 7 5 422 (-46.123421) :167 :106 :8 :167 :121 :168 :106
61 8 1 11 (-45.378798) :329 :373 :9 :133 :167 :106 :8 :167
61 8 1 147 (-38.966950) :89 :373 :9 :133 :167 :106 :8 :167
61 8 3 120 (-48.844046) :9 :133 :396 :106 :8 :167 :121 :168
61 8 3 5 (-51.119849) :9 :133 :396 :332 :8 :167 :121 :168
61 8 3 58 (-52.147905) :9 :133 :167 :106 :8 :167 :20 :168
61 8 3 110 (-41.618256) :9 :133 :167 :106 :8 :167 :121 :168
61 8 4 34 (-64.510769) :133 :167 :106 :8 :167 :20 :168 :106
61 8 4 342 (-53.875019) :133 :167 :106 :8 :167 :121 :168 :106
61 8 4 109 (-72.548313) :133 :396 :106 :8 :167 :20 :168 :106
61 8 4 644 (-60.362608) :133 :396 :106 :8 :167 :121 :168 :106
61 8 4 19 (-63.582264) :133 :396 :332 :8 :167 :121 :168 :106
61 9 2 6 (-52.198638) :373 :9 :133 :167 :106 :8 :167 :121 :168
61 9 3 240 (-64.021578) :9 :133 :396 :106 :8 :167 :121 :168 :106
61 9 3 10 (-66.297381) :9 :133 :396 :332 :8 :167 :121 :168 :106
61 9 3 28 (-64.744047) :9 :133 :167 :106 :8 :167 :20 :168 :106
61 9 3 282 (-54.214397) :9 :133 :167 :106 :8 :167 :121 :168 :106
61 10 1 11 (-56.125037) :329 :373 :9 :133 :167 :106 :8 :167 :121 :168
61 10 1 79 (-60.242839) :89 :373 :9 :133 :167 :106 :8 :167 :20 :168
61 10 1 147 (-49.713189) :89 :373 :9 :133 :167 :106 :8 :167 :121 :168
61 10 1 288 (-55.791899) :89 :373 :9 :133 :396 :106 :8 :167 :121 :168
61 10 1 12 (-58.067702) :89 :373 :9 :133 :396 :332 :8 :167 :121 :168
61 10 2 12 (-67.376170) :373 :9 :133 :167 :106 :8 :167 :121 :168 :106
61 11 1 33 (-69.866798) :329 :373 :9 :133 :167 :106 :8 :167 :121 :168 :106
61 11 1 56 (-72.714732) :89 :373 :9 :133 :167 :106 :8 :167 :20 :168 :106
61 11 1 417 (-62.185083) :89 :373 :9 :133 :167 :106 :8 :167 :121 :168 :106
61 11 1 840 (-68.263793) :89 :373 :9 :133 :396 :106 :8 :167 :121 :168 :106
61 11 1 35 (-70.539596) :89 :373 :9 :133 :396 :332 :8 :167 :121 :168 :106
Ref :89 :373 :9 :133 :167 :106 :8 :167 :121 :168 :106 :171
```

Forest and reference for `trace_np_sample_7` (the 19-word sentence in figures 3.15 and 3.16):

```
566 1 1 1 (-1.223774) :66
566 1 1 1 (-2.196704) :8
566 1 2 24 (-7.590600) :164
566 1 3 1 (-0.737931) :108
566 1 3 1 (-0.329753) :111
566 1 3 1 (0.000000) :115
566 1 3 1 (-0.005837) :121
566 1 3 1 (-5.295815) :20
566 1 3 1 (0.000000) :323
566 1 3 1 (-1.981002) :335
566 1 3 1 (-3.772761) :336
566 1 4 22 (-5.665932) :167
566 1 5 1 (-0.000436) :408
566 1 6 3 (-6.893990) :392
566 1 6 12 (-5.260332) :400
566 1 7 1 (-2.905078) :108
566 1 7 1 (-2.542726) :109
566 1 7 1 (-1.780588) :111
566 1 7 1 (-0.116972) :122
566 1 7 1 (0.000000) :124
566 1 8 9 (-5.464773) :168
566 1 9 3 (-0.033056) :14
566 1 9 1 (0.000000) :292
566 1 9 1 (-0.693147) :338
566 1 9 1 (0.000000) :61
566 1 10 9 (-5.464773) :168
566 1 11 1 (-0.000436) :408
566 1 12 1 (-1.531477) :304
566 1 12 3 (-0.114786) :373
566 1 13 12 (-6.347391) :391
566 1 13 13 (-6.677080) :399
566 1 14 1 (-1.661885) :108
566 1 14 1 (-2.186047) :111
566 1 14 1 (-0.011651) :119
566 1 14 1 (-0.154151) :320
566 1 14 8 (-5.934895) :325
566 1 14 1 (0.000000) :342
566 1 14 3 (-0.015164) :362
566 1 15 13 (-4.605910) :389
566 1 16 1 (-4.204691) :425
```

```
566 1 16 1 (0.000000) :54
566 1 16 1 (-0.054068) :64
566 1 16 1 (-0.083382) :68
566 1 16 1 (-0.405465) :70
566 1 16 1 (-0.133531) :86
566 1 16 1 (-3.881562) :91
566 1 16 5 (-4.327440) :96
566 1 16 1 (-0.205020) :9
566 1 17 22 (-6.513230) :167
566 1 18 1 (-0.038578) :105
566 1 18 1 (-3.310542) :108
566 1 18 1 (-0.475423) :303
566 1 18 1 (-5.159055) :335
566 1 18 1 (-0.530629) :33
566 1 19 4 (-3.931828) :128
566 1 19 22 (-7.611842) :167
566 1 19 18 (-4.688521) :396
566 2 1 31 (-9.787303) :8 :164
566 2 3 2 (-12.942748) :20 :167
566 2 3 12 (-8.069664) :121 :167
566 2 7 8 (-10.081556) :122 :168
566 2 9 4 (-5.622993) :14 :168
566 2 12 1 (-7.512412) :373 :399
566 2 13 3 (-18.476517) :399 :325
566 2 13 11 (-20.995347) :391 :362
566 2 13 5 (-20.022174) :391 :325
566 2 14 6 (-5.084739) :362 :389
566 2 15 4 (-11.133143) :389 :96
566 2 16 9 (-10.010996) :9 :167
566 2 18 33 (-11.477082) :105 :167
566 2 18 8 (-9.043879) :105 :396
566 3 2 6 (-23.451118) :164 :20 :167
566 3 2 40 (-18.934832) :164 :121 :167
566 3 4 3 (-25.983215) :167 :408 :400
566 3 6 9 (-17.792572) :400 :122 :168
566 3 6 2 (-20.404650) :392 :122 :168
566 3 8 16 (-11.908747) :168 :14 :168
566 3 13 1 (-20.872040) :399 :362 :389
566 3 13 12 (-23.713547) :391 :362 :389
566 3 14 6 (-14.971460) :362 :389 :96
566 3 15 10 (-16.726888) :389 :9 :167
566 3 17 16 (-21.689190) :167 :105 :396
566 3 17 64 (-23.921937) :167 :105 :167
566 4 1 95 (-18.167553) :8 :164 :121 :167
566 4 1 14 (-28.035242) :8 :164 :20 :167
566 4 3 5 (-25.989052) :121 :167 :408 :400
566 4 7 29 (-17.218674) :122 :168 :14 :168
566 4 13 1 (-27.308774) :399 :362 :389 :96
566 4 13 12 (-32.573799) :391 :362 :389 :96
566 4 14 12 (-16.723576) :362 :389 :9 :167
566 4 16 8 (-22.365489) :9 :167 :105 :396
566 4 16 30 (-24.598236) :9 :167 :105 :167
566 5 2 31 (-35.189517) :164 :121 :167 :408 :400
566 5 4 6 (-37.451063) :167 :408 :400 :122 :168
566 5 6 31 (-25.769892) :400 :122 :168 :14 :168
566 5 6 6 (-30.829638) :392 :122 :168 :14 :168
566 5 13 2 (-28.899959) :399 :362 :389 :9 :167
566 5 13 35 (-34.164984) :391 :362 :389 :9 :167
566 5 15 52 (-30.649244) :389 :9 :167 :105 :167
566 5 15 10 (-28.416497) :389 :9 :167 :105 :396
566 6 1 78 (-34.998480) :8 :164 :121 :167 :408 :400
566 6 3 10 (-37.456900) :121 :167 :408 :400 :122 :168
566 6 14 15 (-28.413185) :362 :389 :9 :167 :105 :396
566 6 14 55 (-30.645932) :362 :389 :9 :167 :105 :167
566 7 2 62 (-46.657366) :164 :121 :167 :408 :400 :122 :168
566 7 4 24 (-44.588182) :167 :408 :400 :122 :168 :14 :168
566 7 13 5 (-40.589568) :399 :362 :389 :9 :167 :105 :396
566 7 13 16 (-42.822315) :399 :362 :389 :9 :167 :105 :167
566 7 13 38 (-45.854593) :391 :362 :389 :9 :167 :105 :396
566 7 13 126 (-48.087340) :391 :362 :389 :9 :167 :105 :167
566 8 1 170 (-46.466328) :8 :164 :121 :167 :408 :400 :122 :168
566 8 3 40 (-44.594019) :121 :167 :408 :400 :122 :168 :14 :168
566 9 2 248 (-53.794484) :164 :121 :167 :408 :400 :122 :168 :14 :168
```

```
566 10 1 674 (-53.603447) :8 :164 :121 :167 :408 :400 :122 :168 :14 :168
Ref :8 :164 :121 :167 :408 :400 :122 :168 :14 :168 :408 :373 :399 :362 :389 :9 :167 :105 :396
```

Forest and reference for `trace_np_sample_8` (the 25-word sentence in figures 3.17 and 3.18):

```
921 1 1 1 (-0.014512) :414
921 1 2 1 (-0.924948) :156
921 1 2 1 (-1.386294) :234
921 1 2 3 (0.000000) :275
921 1 2 1 (-4.234103) :97
921 1 3 5 (-3.000892) :380
921 1 3 3 (-3.874669) :385
921 1 4 12 (-7.932960) :133
921 1 4 13 (-3.744494) :334
921 1 5 3 (-4.329047) :392
921 1 5 12 (-5.578786) :400
921 1 6 8 (-2.178343) :106
921 1 6 1 (-5.613127) :107
921 1 6 1 (-3.390023) :109
921 1 6 1 (0.000000) :112
921 1 6 1 (-0.405465) :294
921 1 6 7 (-2.510691) :332
921 1 7 1 (0.000000) :113
921 1 7 1 (-1.098613) :338
921 1 7 1 (0.000000) :62
921 1 7 1 (-0.348307) :66
921 1 7 1 (-0.120895) :8
921 1 8 9 (-6.968850) :168
921 1 9 8 (-3.688879) :348
921 1 9 39 (-1.386294) :349
921 1 9 3 (-0.386773) :38
921 1 10 5 (-0.921683) :329
921 1 10 1 (-0.483871) :89
921 1 11 4 (-0.006509) :368
921 1 12 1 (-4.204691) :425
921 1 12 1 (0.000000) :54
921 1 12 1 (-0.054068) :64
921 1 12 1 (-0.083382) :68
921 1 12 1 (-0.405465) :70
921 1 12 1 (-0.133531) :86
921 1 12 1 (-3.881562) :91
921 1 12 5 (-4.327440) :96
921 1 12 1 (-0.205020) :9
921 1 13 12 (-5.448052) :133
921 1 14 24 (-8.283747) :164
921 1 15 3 (-0.033056) :14
921 1 15 1 (0.000000) :292
921 1 15 1 (-0.693147) :338
921 1 15 1 (0.000000) :61
921 1 16 1 (-0.924948) :156
921 1 16 1 (-1.386294) :234
921 1 16 3 (0.000000) :275
921 1 16 1 (-4.234103) :97
921 1 17 4 (-0.006509) :368
921 1 18 15 (-3.951242) :394
921 1 19 8 (-2.178343) :106
921 1 19 1 (-5.613127) :107
921 1 19 1 (-3.390023) :109
921 1 19 1 (0.000000) :112
921 1 19 1 (-0.405465) :294
921 1 19 7 (-2.510691) :332
921 1 20 1 (0.000000) :113
921 1 20 1 (-1.098613) :338
921 1 20 1 (0.000000) :62
921 1 20 1 (-0.348307) :66
921 1 20 1 (-0.120895) :8
921 1 21 24 (-8.283747) :164
921 1 22 1 (-0.013841) :415
921 1 23 1 (-0.000436) :408
921 1 24 5 (-0.435517) :271
921 1 25 3 (-2.060361) :392
921 1 25 12 (-4.613706) :400
921 2 3 1 (-18.660845) :380 :334
```

```
921 2 4 2 (-19.613414) :133 :400
921 2 4 12 (-11.697876) :334 :400
921 2 5 2 (-10.958428) :400 :332
921 2 5 7 (-14.707623) :400 :106
921 2 5 2 (-14.436304) :392 :106
921 2 5 2 (-12.327628) :392 :332
921 2 7 11 (-11.463931) :8 :168
921 2 12 1 (-6.346220) :9 :133
921 2 13 9 (-15.118094) :133 :164
921 2 15 1 (-0.958004) :14 :156
921 2 15 1 (-0.726204) :14 :275
921 2 17 1 (-6.553006) :368 :394
921 2 18 6 (-13.322711) :394 :332
921 2 18 12 (-15.699652) :394 :106
921 2 20 31 (-8.404642) :8 :164
921 2 24 11 (-11.313155) :271 :392
921 3 2 11 (-26.982001) :275 :380 :133
921 3 4 1 (-21.646699) :334 :400 :106
921 3 6 34 (-15.171871) :106 :8 :168
921 3 10 11 (-14.170294) :89 :368 :96
921 3 12 11 (-16.501768) :9 :133 :164
921 3 19 76 (-13.668369) :106 :8 :164
921 4 2 11 (-47.815877) :275 :380 :133 :400
921 4 5 10 (-22.628556) :392 :106 :8 :168
921 4 5 42 (-22.899875) :400 :106 :8 :168
921 4 5 1 (-24.549828) :392 :332 :8 :168
921 4 16 11 (-23.287258) :275 :368 :394 :106
921 4 18 170 (-22.388403) :394 :106 :8 :164
921 4 18 5 (-22.859773) :394 :332 :8 :164
921 5 4 10 (-28.923942) :334 :400 :106 :8 :168
921 5 10 12 (-24.447746) :89 :368 :9 :133 :164
921 5 21 33 (-25.148513) :164 :415 :408 :271 :392
921 6 16 55 (-29.976008) :275 :368 :394 :106 :8 :164
921 6 20 26 (-25.987129) :8 :164 :415 :408 :271 :392
921 7 19 97 (-30.578780) :106 :8 :164 :415 :408 :271 :392
921 8 18 102 (-48.261014) :394 :106 :8 :164 :415 :408 :271 :392
921 10 16 88 (-42.113387) :275 :368 :394 :106 :8 :164 :415 :408 :271 :392
Ref :414 :275 :380 :334 :400 :332 :8 :168 :38 :89 :368 :9 :133 :164 :14 :275 :368 :394 :106 :8 :164 :415 :408 :271 :392
```

Appendix E

PERL scripts and utilities

The following pieces of PERL source code are the main scripts which implement different tasks such as changing text files from one syntax or format to another, or the calculus of features of corpora, lexica, grammars, forests, ...

E.1 susanne_treebank_to_tagged_corpus.prl

```
#!/usr/bin/perl
# susanne_treebank_to_tagged_corpus.prl
#
# We assume <st> to be a file in the syntax of Susanne treebank.
# This program produces the corresponding tagged corpus.
#
# Example of input:
#
#    Reference Status Tag    Word           Lemma         Parsing
#    --------- ------ ------ -------------- ------------- -------------------------
#    A01:0010b    -   AT     The            the           [O[S[Nns:s.
#    A01:0010c    -   NP1s   Fulton         Fulton        [Nns.
#    A01:0010d    -   NNL1cb County         county        .Nns]
#    A01:0010e    -   JJ     Grand          grand         .
#    A01:0010f    -   NN1c   Jury           jury          .Nns:s]
#    A01:0010g    -   VVDv   said           say           [Vd.Vd]
#    A01:0010h    -   NPD1   Friday         Friday        [Nns:t.Nns:t]
#    A01:0010i    -   AT1    an             an            [Fn:o[Ns:s.
#    A01:0010j    -   NN1n   investigation  investigation .
#    A01:0020a    -   IO     of             of            [Po.
#    A01:0020b    -   NP1t   Atlanta        Atlanta       [Ns[G[Nns.Nns]
#    A01:0020c    -   GG     +<apos>s       -             .G]
#    A01:0020d    -   JJ     recent         recent        .
#    A01:0020e    -   JJ     primary        primary       .
#    A01:0020f    -   NN1n   election       election      .Ns]Po]Ns:s]
#    A01:0020g    -   VVDv   produced       produce       [Vd.Vd]
#    A01:0020h    -   YIL    <ldquo>        -             .
#    A01:0020i    -   ATn    +no            no            [Ns:o.
#    A01:0020j    -   NN1u   evidence       evidence      .
#    A01:0020k    -   YIR    +<rdquo>       -             .
#    A01:0020m    -   CST    that           that          [Fn.
#    A01:0030a    -   DDy    any            any           [Np:s.
#    A01:0030b    -   NN2    irregularities irregularity  .Np:s]
#    A01:0030c    -   VVDv   took           take          [Vd.Vd]
#    A01:0030d    -   NNL1c  place          place         [Ns:o.Ns:o]Fn]Ns:o]Fn:o]S]
#    A01:0030e    -   YF     +.             -             .O]
```

```
#
# Example of output:
#
#    The/AT Fulton/NP1s County/NNL1cb Grand/JJ Jury/NN1c said/VVDv Friday/NPD1
#    an/AT1 investigation/NN1n of/IO Atlanta/NP1t <apos>s/GG recent/JJ primary/JJ
#    election/NN1n produced/VVDv <ldquo>/YIL no/ATn evidence/NN1u <rdquo>/YIR
#    that/CST any/DDy irregularities/NN2 took/VVDv place/NNL1c ./YF
#
# Usage:
#    cat <st> | susanne_treebank_to_tagged_corpus.prl

$first_word = 1;

while ($line = <STDIN>) {
    chomp $line;
    @fields = split("\t", $line);

    $fields[3] =~ s/^\+//;

    @parsing_char_by_char = split(//, $fields[5]);
    foreach $c (@parsing_char_by_char) {
        SWITCH: {
            if ($c eq "[") {
                $depth++;
                last SWITCH;
            }
            if ($c eq "]") {
                $depth--;
                if ($depth == 0) { print "\n"; $first_word = 1; }
                last SWITCH;
            }
            if ($c eq ".") {
                if ($fields[2] ne "YG") { # YG means traced parsing
                    if ($first_word) { $first_word = 0; } else { print " "; }
                    print "$fields[3]/$fields[2]";
                }
                last SWITCH;
            }
        }
    }
}
```

E.2 corpus_to_lexicon.prl

```
#!/usr/bin/perl
# corpus_to_lexicon.prl
#
# We assume <corpus> to be a corpus with lines
#
#    form1/tag1 form2/tag2 ... formn/tagn ./tag.
#
# This program uses <corpus> in order to
# generate a lexicon with lines
#
#    form1 tag1
#    form2 tag2
#    ...
```

```
#    formn tagn
#
# Usage:
#    cat <corpus> | corpus_to_lexicon.prl | sort | tagged_words_to_lexicon.prl

while ($line = <STDIN>) {
    chomp $line;
    @words = split(" ", $line);
    foreach $w (@words) {
        ($form, $tag) = split("/", $w);
        print "$form $tag\n";
    }
}
```

E.3 tagged_words_to_lexicon.prl

```
#!/usr/bin/perl
# tagged_words_to_lexicon.prl
#
# We assume <lex1> <lex2> ... <lexn> to be files with lines
#
#    form tag1 tag2 ... tagn
#
# This program replaces lines which have the same form with
# only one line which contains the form and the union of the tags,
# sorted by frequency (first one the most frequent and last one
# the least frequent).
#
# Usage:
#    cat <lex1> <lex2> ... <lexn> | sort | tagged_words_to_lexicon.prl

sub numeric_sort { $a <=> $b }

sub sort_tags {
    my %tag_occurrencies;
    my %occurrencies_tag;
    my $tags;

    foreach $t (@_) {
        $tag_occurrencies{$t}++;
    }
    foreach $t (keys %tag_occurrencies) {
        $occurrencies_tag{$tag_occurrencies{$t}} .= " $t";
    }
    foreach $n (sort numeric_sort keys %occurrencies_tag) {
        $tags = $occurrencies_tag{$n} . $tags;
    }
    return ($tags);
}

$line = <STDIN>;
@data = split(" ", $line);
$old_form = $data[0];
shift(@data);
$old_tags = join(" ", @data);

while ($line = <STDIN>) {
```

```
   @data = split(" ", $line);
   $form = $data[0];
   shift(@data);
   $tags = join(" ", @data);

   if ($form eq $old_form) {
       $old_tags .= " $tags";
   } else {
       @union_tags = sort_tags(split(" ", $old_tags));
       print "$old_form@union_tags\n";
       $old_form = $form;
       $old_tags = $tags;
   }
}

@union_tags = sort_tags(split(" ", $old_tags));
print "$old_form@union_tags\n";
```

E.4 features_lexicon.prl

```
#!/usr/bin/perl
# features_lexicon.prl
#
# We assume <lex> to be a lexicon with lines
#
#    form tag1 tag2 ... tagn
#
# This program calculates the number of lines and provides
# statistics about the number of ambiguities in them.
#
# Usage:
#    cat <lex> | features_lexicon.prl

sub numeric_sort { $a <=> $b }

while ($line = <STDIN>) {
    $n++;
    @data = split(" ", $line);
    $ambiguities{$#data}++;
}

foreach $num (sort numeric_sort keys %ambiguities) {
    $p += $num * $ambiguities{$num};
    print "*with $num tags: $ambiguities{$num} forms\n";
}

print "\nlexicon size: $n forms, $p tags\n";
```

E.5 features_corpus.prl

```
#!/usr/bin/perl
# features_corpus.prl
#
# We assume <lex> to be a lexicon with lines
#
#    form tag1 tag2 ... tagn
```

```
#
# and <corpus> to be a corpus with lines
#
#    form1/tag1 form2/tag2 ... formn/tagn ./tag.
#
#    (lines could also end with ?/tag? or .../tag... or another
#     punctuation mark, and thus each line is a sentence)
#
# This program calculates the number of words in <corpus> and provides
# statistics about the number of ambiguities in them, in regard to <lex>.
#
# Usage:
#    features_corpus.prl <lex> <corpus>

sub numeric_sort { $a <=> $b }

($#ARGV + 1 == 2) or die "\nUsage: $0 <lex> <corpus>\n\n";
open (LL, "$ARGV[0]") or die "Can't open $ARGV[0]: $!\n";
open (XX, "$ARGV[1]") or die "Can't open $ARGV[1]: $!\n";

print STDERR "Loading lexicon...\n";
while ($lineLL = <LL>) {
    @fts = split (" ", $lineLL);
    $lexicon{$fts[0]} = $#fts;
}

print STDERR "Calculating features...\n";

while ($lineXX = <XX>) {
    @wordsXX = split(" ", $lineXX);

    foreach $i (0..$#wordsXX) {
        $n++;
        ($formXX, $tagXX) = split ("/", $wordsXX[$i]);
        $features{$lexicon{$formXX}}++;
    }
}

foreach $num (sort numeric_sort keys %features) {
    $p += $num * $features{$num};
    print "*with $num tags: $features{$num} words\n";
}

print "\nlexicon size: $n words, $p tags\n";
```

E.6 gen_slptoolkit_tag_table.prl

```
#!/usr/bin/perl
# gen_slptoolkit_tag_table.prl
#
# We assume <tag-set> to be a file with lines
#
#    tag1
#    tag2
#    ...
#    tagn
#
```

```
# This program generates lines
#
#    1:tag1
#    2:tag2
#    ...
#    n:tagn
#
# Usage:
#    cat <tag-set> | gen_slptoolkit_tag_table.prl

$n = 1;
while ($line = <STDIN>) {
    print "$n:$line";
    $n++;
}
```

E.7 lexicon_to_rules.prl

```
#!/usr/bin/perl
# lexicon_to_rules.prl
#
# We assume <lex> to be a lexicon with lines
#
#    form tag1 tag2 ... tagn
#
# This program generates rules
#
#    tag1 -> form
#    tag2 -> form
#    ...
#    tagn -> form
#
# Usage:
#    cat <lex> | lexicon_to_rules.prl

while ($line = <STDIN>) {
    chomp $line;
    @tags = split (" ", $line);
    $form = shift(@tags);

    foreach $t (@tags) {
        print "$t -> $form\n";
    }
}
```

E.8 rules_to_stochastic_grammar.prl

```
#!/usr/bin/perl
# rules_to_stochastic_grammar.prl
#
# We assume <rules> to be a file with lines
#
#    NT1 -> NT2 NT3 ... NTn
#
# This program replaces lines which are equal with only one line
#
```

```
#    NT1 -> NT2 NT3 ... NTn (p)
#
# where p is the probability of the rule, calculated as
#
#                   number of ocurrencies of the rule
#    ------------------------------------------------------------------
#    number of rules which have the same non-terminal NT1 to the left
#
# Usage:
#    cat <rules> | sort | rules_to_stochastic_grammar.prl

@list_of_rules = ();
@list_of_occurrencies = ();

$rule = <STDIN>;
chomp $rule;
$number_of_rules = 1;
$occurrencies_of_this_rule = 1;
@data = split(" ", $rule);
$left_hand_side = $data[0];

while ($new_rule = <STDIN>) {
    chomp $new_rule;
    if ($new_rule eq $rule) {
        $number_of_rules++;
        $occurrencies_of_this_rule++;
    } else {
        @data = split(" ", $new_rule);
        $new_left_hand_side = $data[0];
        if ($new_left_hand_side eq $left_hand_side) {
            $number_of_rules++;
            push (@list_of_rules, $rule);
            push (@list_of_occurrencies, $occurrencies_of_this_rule);
            $rule = $new_rule;
            $occurrencies_of_this_rule = 1;
        } else {
            push (@list_of_rules, $rule);
            push (@list_of_occurrencies, $occurrencies_of_this_rule);
            foreach $i (0..$#list_of_rules) {
                $probability = $list_of_occurrencies[$i]/$number_of_rules;
                print "$list_of_rules[$i] ";
                printf "(%lg)\n", $probability;
            }
            @list_of_rules = ();
            @list_of_occurrencies = ();
            $rule = $new_rule;
            $left_hand_side = $new_left_hand_side;
            $number_of_rules = 1;
            $occurrencies_of_this_rule = 1;
        }
    }
}

push (@list_of_rules, $rule);
push (@list_of_occurrencies, $occurrencies_of_this_rule);
foreach $i (0..$#list_of_rules) {
    $probability = $list_of_occurrencies[$i]/$number_of_rules;
    print "$list_of_rules[$i] ";
```

```
    printf "(%lg)\n", $probability;
}
```

E.9 susanne_grammar_to_slptoolkit_lexicon.prl

```
#!/usr/bin/perl
# susanne_grammar_to_slptoolkit_lexicon.prl
#
# We assume <tag-table> to be a file with lines
#
#    1:tag1
#    2:tag2
#    ...
#    n:tagn
#
# and <grammar> to be a files with rules
#
#    TAG -> form (probability)
#
# This program translates <grammar> into a lexicon in SLPToolKit syntax.
#
# Example: (we assume line "1:APPGf" belongs to <tag-table>)
#
#    old rule:   APPGf -> Her (0.5)
#    new rule:   Her||1||0.5
#
# Usage:
#    susanne_grammar_to_slptoolkit_lexicon.prl <tag-table> <grammar> | sort

($#ARGV + 1 == 2) or die "\nUsage: $0 <tag-table> <grammar>\n\n";
open (TT, "$ARGV[0]") or die "Can't open $ARGV[0]: $!\n";
open (SG, "$ARGV[1]") or die "Can't open $ARGV[1]: $!\n";

print STDERR "Loading tag table...\n";
while ($lineTT = <TT>) {
    chomp $lineTT;
    ($n, $t) = split (":", $lineTT);
    $tag_table{$t} = $n;
}

print STDERR "Calculating new lexicon...\n";
while ($lineSG = <SG>) {
    chomp $lineSG;
    ($tag, $arrow, $form, $probability) = split (" ", $lineSG);
    print "$form||";

    $n = $tag_table{$tag};
    if ($n ne "") {
        print "$n";
    } else {
        print STDERR "error: $tag is not in tag-table $ARGV[0]";
        exit;
    }

    $probability =~ s/\(//g;
    $probability =~ s/\)//g;
    print "||$probability\n";
```

```
}

print STDERR "Done...\n";
```

E.10 susanne_treebank_to_susanne_parsing.prl

```
#!/usr/bin/perl
# susanne_treebank_to_susanne_parsing.prl
#
# We assume <st> to be a file in the syntax of Susanne treebank.
# This program produces the corresponding parsing.
#
# Example of input:
#
#    Reference Status Tag    Word           Lemma         Parsing
#    --------- ------ ------ -------------- ------------- -------------------------
#    A01:0010b    -   AT     The            the           [O[S[Nns:s.
#    A01:0010c    -   NP1s   Fulton         Fulton        [Nns.
#    A01:0010d    -   NNL1cb County         county        .Nns]
#    A01:0010e    -   JJ     Grand          grand         .
#    A01:0010f    -   NN1c   Jury           jury          .Nns:s]
#    A01:0010g    -   VVDv   said           say           [Vd.Vd]
#    A01:0010h    -   NPD1   Friday         Friday        [Nns:t.Nns:t]
#    A01:0010i    -   AT1    an             an            [Fn:o[Ns:s.
#    A01:0010j    -   NN1n   investigation  investigation .
#    A01:0020a    -   IO     of             of            [Po.
#    A01:0020b    -   NP1t   Atlanta        Atlanta       [Ns[G[Nns.Nns]
#    A01:0020c    -   GG     +<apos>s       -             .G]
#    A01:0020d    -   JJ     recent         recent        .
#    A01:0020e    -   JJ     primary        primary       .
#    A01:0020f    -   NN1n   election       election      .Ns]Po]Ns:s]
#    A01:0020g    -   VVDv   produced       produce       [Vd.Vd]
#    A01:0020h    -   YIL    <ldquo>        -             .
#    A01:0020i    -   ATn    +no            no            [Ns:o.
#    A01:0020j    -   NN1u   evidence       evidence      .
#    A01:0020k    -   YIR    +<rdquo>       -             .
#    A01:0020m    -   CST    that           that          [Fn.
#    A01:0030a    -   DDy    any            any           [Np:s.
#    A01:0030b    -   NN2    irregularities irregularity  .Np:s]
#    A01:0030c    -   VVDv   took           take          [Vd.Vd]
#    A01:0030d    -   NNL1c  place          place         [Ns:o.Ns:o]Fn]Ns:o]Fn:o]S]
#    A01:0030e    -   YF     +.             -             .O]
#
# Example of output:
#
#    [ O [ S [ Nns:s [ AT The AT ] [ Nns [ NP1s Fulton NP1s ] [ NNL1cb County
#    NNL1cb ] Nns ] [ JJ Grand JJ ] [ NN1c Jury NN1c ] Nns:s ] [ Vd [ VVDv
#    said VVDv ] Vd ] [ Nns:t [ NPD1 Friday NPD1 ] Nns:t ] [ Fn:o [ Ns:s
#    [ AT1 an AT1 ] [ NN1n investigation NN1n ] [ Po [ IO of IO ] [ Ns [ G
#    [ Nns [ NP1t Atlanta NP1t ] Nns ] [ GG <apos>s GG ] G ] [ JJ recent JJ ]
#    [ JJ primary JJ ] [ NN1n election NN1n ] Ns ] Po ] Ns:s ] [ Vd [ VVDv
#    produced VVDv ] Vd ] [ YIL <ldquo> YIL ] [ Ns:o [ ATn no ATn ] [ NN1u
#    evidence NN1u ] [ YIR <rdquo> YIR ] [ Fn [ CST that CST ] [ Np:s [ DDy
#    any DDy ] [ NN2 irregularities NN2 ] Np:s ] [ Vd [ VVDv took VVDv ] Vd ]
#    [ Ns:o [ NNL1c place NNL1c ] Ns:o ] Fn ] Ns:o ] Fn:o ] S ] [ YF . YF ] O ]
#
# Usage:
```

```
#    cat <st> | susanne_treebank_to_susanne_parsing.prl

$blank_printed = 1;

while ($line = <STDIN>) {
    chomp $line;
    @fields = split("\t", $line);

    $fields[3] =~ s/^\+//;

    @parsing_char_by_char = split(//, $fields[5]);
    foreach $c (@parsing_char_by_char) {
        SWITCH: {
            if ($c eq "[") {
                $depth++;
                if (!$blank_printed) { print " "; }
                print "[ ";
                $blank_printed = 1;
                last SWITCH;
            }
            if ($c eq "]") {
                $depth--;
                if ($depth == 0) {
                    print " ]\n";
                } else {
                    print " ] ";
                }
                $blank_printed = 1;
                last SWITCH;
            }
            if ($c eq ".") {
                if (!$blank_printed) { print " "; }
                print "[ $fields[2] $fields[3] $fields[2] ] ";
                $blank_printed = 1;
                last SWITCH;
            }

            print "$c"; $blank_printed = 0;
        }
    }
}
```

E.11 extract_real_sentences.prl

```
#!/usr/bin/perl
# extract_real_sentences.prl
#
# We assume <tag-table> to be a file with lines
#
#    1:tag1
#    2:tag2
#    ...
#    n:tagn
#
# and <parsing> to be a file in the syntax of Susanne parsing.
#
# This program takes each line of <parsing> and removes all that is
```

```
# necessary (paragraph or title marks, for instance) in order to keep
# only those subparsings that correspond to real sentences, that is,
# subparsings with a root non-terminal starting with S, F, T, W, A, Z
# or L.
#
# Example of input:
#
#    [ O [ S [ Nns:s [ AT The AT ] [ Nns [ NP1s Fulton NP1s ] [ NNL1cb County
#    NNL1cb ] Nns ] [ JJ Grand JJ ] [ NN1c Jury NN1c ] Nns:s ] [ Vd [ VVDv
#    said VVDv ] Vd ] [ Nns:t [ NPD1 Friday NPD1 ] Nns:t ] [ Fn:o [ Ns:s
#    [ AT1 an AT1 ] [ NN1n investigation NN1n ] [ Po [ IO of IO ] [ Ns [ G
#    [ Nns [ NP1t Atlanta NP1t ] Nns ] [ GG <apos>s GG ] G ] [ JJ recent JJ ]
#    [ JJ primary JJ ] [ NN1n election NN1n ] Ns ] Po ] Ns:s ] [ Vd [ VVDv
#    produced VVDv ] Vd ] [ YIL <ldquo> YIL ] [ Ns:o [ ATn no ATn ] [ NN1u
#    evidence NN1u ] [ YIR <rdquo> YIR ] [ Fn [ CST that CST ] [ Np:s [ DDy
#    any DDy ] [ NN2 irregularities NN2 ] Np:s ] [ Vd [ VVDv took VVDv ] Vd ]
#    [ Ns:o [ NNL1c place NNL1c ] Ns:o ] Fn ] Ns:o ] Fn:o ] S ] [ YF . YF ] O ]
#
# Example of output:
#
#    [ S [ Nns:s [ AT The AT ] [ Nns [ NP1s Fulton NP1s ] [ NNL1cb County
#    NNL1cb ] Nns ] [ JJ Grand JJ ] [ NN1c Jury NN1c ] Nns:s ] [ Vd [ VVDv
#    said VVDv ] Vd ] [ Nns:t [ NPD1 Friday NPD1 ] Nns:t ] [ Fn:o [ Ns:s
#    [ AT1 an AT1 ] [ NN1n investigation NN1n ] [ Po [ IO of IO ] [ Ns [ G
#    [ Nns [ NP1t Atlanta NP1t ] Nns ] [ GG <apos>s GG ] G ] [ JJ recent JJ ]
#    [ JJ primary JJ ] [ NN1n election NN1n ] Ns ] Po ] Ns:s ] [ Vd [ VVDv
#    produced VVDv ] Vd ] [ YIL <ldquo> YIL ] [ Ns:o [ ATn no ATn ] [ NN1u
#    evidence NN1u ] [ YIR <rdquo> YIR ] [ Fn [ CST that CST ] [ Np:s [ DDy
#    any DDy ] [ NN2 irregularities NN2 ] Np:s ] [ Vd [ VVDv took VVDv ] Vd ]
#    [ Ns:o [ NNL1c place NNL1c ] Ns:o ] Fn ] Ns:o ] Fn:o ] S ]
#
# Usage:
#    extract_real_sentences.prl <tag-table> <parsing>

($#ARGV + 1 == 2) or die "\nUsage: $0 <tag-table> <parsing>\n\n";
open (TT, "$ARGV[0]") or die "Can't open $ARGV[0]: $!\n";
open (PS, "$ARGV[1]") or die "Can't open $ARGV[1]: $!\n";

sub is_clause_tag {
    my $t = shift(@_);
    if ($tag_table{$t} ne "") { return 0; }
    if ($t =~ /^S/) { return 1; }
    if ($t =~ /^F/) { return 1; }
    if ($t =~ /^T/) { return 1; }
    if ($t =~ /^W/) { return 1; }
    if ($t =~ /^A/) { return 1; }
    if ($t =~ /^Z/) { return 1; }
    if ($t =~ /^L/) { return 1; }
    return 0;
}

sub split_in_sentences {
    my $parsing = shift(@_);
    my @symbols = split (" ", $parsing);
    my $sentence = "";
    my @sentences = ();
    my $depth = 0;
    my $i = 0;
```

```
    my $begin = 0;
    my $end = 0;

    while ($i <= $#symbols) {
        if ($symbols[$i] eq "[") {
            # treatment for sequence "[ NT"
            if (($symbols[$i+1] eq "YPL") or ($symbols[$i+1] eq "YPR")) {
                # special treatment for sequences "[ YPL [ YPL ]" and "[ YPR ] YPR ]"
                $i += 5;
            } else {
                # normal processing
                $depth++;
                $i += 2;
            }
        } else {
            if ($symbols[$i] eq "]") {
                $depth--;
            }
            $i++;
        }

        if ($depth == 0) {
            $end = $i - 1;
            $sentence = join (" ", @symbols[$begin..$end]);
            push (@sentences, $sentence);
            $begin = $i;
        }
    }

    return @sentences;
}

sub flat_sentence {
    my $s = shift(@_);
    my @data = split (" ", $s);
    if (is_clause_tag ($data[1])) {
        print "$s\n";
    } else {
        shift @data; shift @data; pop @data; pop @data;
        if ($#data > 0) {
            $s = join (" ", @data);
            @s = split_in_sentences ($s);
            foreach $e (@s) {
                flat_sentence ($e);
            }
        }
    }
}

print STDERR "Loading tag table...\n";
while ($lineTT = <TT>) {
    chomp $lineTT;
    ($n, $t) = split (":", $lineTT);
    $tag_table{$t} = $n;
}

print STDERR "Calculating new parsing...\n";
```

```
while ($line = <PS>) {
    chomp $line;
    flat_sentence ($line);
}

print STDERR "Done...\n";
```

E.12 divide_susanne_parsing_into_non_traced_and_traced.prl

```
#!/usr/bin/perl
# divide_susanne_parsing_into_non_traced_and_traced.prl
#
# We assume <sp> to be a file in the syntax of Susanne parsing. This
# program splits <sp> in two parts and produces <spnt> which
# contains sentences with no trace, and <spt> which contains
# sentences with traces, that is, with YG marks.
#
# Usage:
#    divide_susanne_parsing_into_non_traced_and_traced.prl <sp> <spnt> <spt>

($#ARGV + 1 == 3) or
    die "\nUsage: $0 <parsing> <non-traced sentences> <traced sentences>\n\n";
open (PARSING, "$ARGV[0]") or die "Can't open $ARGV[0]: $!\n";
open (NON_TRACED, ">$ARGV[1]") or die "Can't open $ARGV[1]: $!\n";
open (TRACED, ">$ARGV[2]") or die "Can't open $ARGV[2]: $!\n";

sub traced_sentence {
    my $s = shift(@_);
    my @data = split(" ", $s);
    foreach $d (@data) {
        if ($d eq "YG") { return 1; } # YG means traced parsing
    }
    return 0;
}

while ($line = <PARSING>) {
    if (traced_sentence ($line)) {
        print TRACED "$line";
    } else {
        print NON_TRACED "$line";
    }
}

close(PARSING);
close(NON_TRACED);
close(TRACED);
```

E.13 susanne_parsing_notrace_to_slptoolkit_parsing.prl

```
#!/usr/bin/perl
# susanne_parsing_notrace_to_slptoolkit_parsing.prl
#
# We assume <tag-table> to be a file with lines
#
#    1:tag1
#    2:tag2
```

```
#    ...
#    n:tagn
#
# and <spnt> to be a file in the syntax of Susanne parsing with no traces.
# This program produces the corresponding parsing in Slptoolkit fromat.
#
# Example of <spnt> (input):
#
#    [ O [ S [ Nns:s [ AT The AT ] [ Nns [ NP1s Fulton NP1s ] [ NNL1cb County
#    NNL1cb ] Nns ] [ JJ Grand JJ ] [ NN1c Jury NN1c ] Nns:s ] [ Vd [ VVDv
#    said VVDv ] Vd ] [ Nns:t [ NPD1 Friday NPD1 ] Nns:t ] [ Fn:o [ Ns:s
#    [ AT1 an AT1 ] [ NN1n investigation NN1n ] [ Po [ IO of IO ] [ Ns [ G
#    [ Nns [ NP1t Atlanta NP1t ] Nns ] [ GG <apos>s GG ] G ] [ JJ recent JJ ]
#    [ JJ primary JJ ] [ NN1n election NN1n ] Ns ] Po ] Ns:s ] [ Vd [ VVDv
#    produced VVDv ] Vd ] [ YIL <ldquo> YIL ] [ Ns:o [ ATn no ATn ] [ NN1u
#    evidence NN1u ] [ YIR <rdquo> YIR ] [ Fn [ CST that CST ] [ Np:s [ DDy
#    any DDy ] [ NN2 irregularities NN2 ] Np:s ] [ Vd [ VVDv took VVDv ] Vd ]
#    [ Ns:o [ NNL1c place NNL1c ] Ns:o ] Fn ] Ns:o ] Fn:o ] S ] [ YF . YF ] O ]
#
# Example of output:
#
#    [ X [ O [ S [ Nns_s [ :8 The ] [ Nns [ :235 Fulton ] [ :181 County ] ] [
#    :133 Grand ] [ :164 Jury ] ] [ Vd [ :392 said ] ] [ Nns_t [ :246 Friday ] ]
#    [ Fn_o [ Ns_s [ :9 an ] [ :167 investigation ] [ Po [ :121 of ] [ Ns [ G
#    [ Nns [ :236 Atlanta ] ] [ :100 <apos>s ] ] [ :133 recent ] [ :133 primary ]
#    [ :167 election ] ] ] ] [ Vd [ :392 produced ] ] [ :414 <ldquo> ] [ Ns_o [
#    :11 no ] [ :168 evidence ] [ :415 <rdquo> ] [ Fn [ :29 that ] [ Np_s [ :88
#    any ] [ :172 irregularities ] ] [ Vd [ :392 took ] ] [ Ns_o [ :180 place ]
#    ] ] ] ] ] [ :411 . ] ] ]
#
# Usage:
#    susanne_parsing_notrace_to_slptoolkit_parsing.prl <tag-table> <spnt>

($#ARGV + 1 == 2) or die "\nUsage: $0 <tag-table> <parsing>\n\n";
open (TT, "$ARGV[0]") or die "Can't open $ARGV[0]: $!\n";
open (SP, "$ARGV[1]") or die "Can't open $ARGV[1]: $!\n";

print STDERR "Loading tag table...\n";
while ($lineTT = <TT>) {
    chomp $lineTT;
    ($n, $t) = split (":", $lineTT);
    $tag_table{$t} = $n;
}

print STDERR "Calculating new parsing...\n";
while ($lineSP = <SP>) {
    chomp $lineSP;
    @data = split(" ", $lineSP);

    print "[ X ";

    $i = 0;
    while ($i <= $#data) {
        SWITCH: {
            if ($data[$i] eq "[") {
                # treatment for sequence "[ NT"
                if (($data[$i+1] eq "YPL") or ($data[$i+1] eq "YPR")) {
                    # special treatment for sequences "[ YPL [ YPL ]" and "[ YPR ] YPR ]"
```

```
                    $ns = $tag_table{$data[$i+1]};
                    if ($ns eq "") {
                        print STDERR "error: $data[$i] is not in tag-table $ARGV[0]\n";
                        exit;
                    }
                    print "[ :$ns $data[$i+2] ] ";
                    $i += 5;
                } else {
                    # normal processing
                    $ns = $tag_table{$data[$i+1]};
                    if ($ns ne "") {
                        print "[ :$ns ";
                    } else {
                        $data[$i+1] =~ s/:/_/g;
                        $data[$i+1] =~ s/-/\$/g;
                        $data[$i+1] =~ s/\"/\^/g;
                        print "[ $data[$i+1] ";
                    }
                    $depth++;
                    $i += 2;
                }
                last SWITCH;
            }

            if ($data[$i] eq "]") {
                print "] ";
                $depth--;
                $i++;
                last SWITCH;
            }

            if ($data[$i+1] eq "]") {
                # treatment for sequences "NT ]", where NT is the same non-terminal
                # as the one in the left hand side of the rule in the top of the
                # rules stack
                $i++;
            } else {
                # treatment for sequences "word NT ]", where NT is the tag of the word
                $depth--;
                if ($data[$i+1] eq "YG") { # YG means traced parsing
                    print STDERR "error: this parsing contains traces\n";
                    exit;
                }
                $ns = $tag_table{$data[$i+1]};
                if ($ns eq "") {
                    print STDERR "error: $data[$i] is not in tag-table $ARGV[0]\n";
                    exit;
                }
                print "$data[$i] ] ";
                $i += 3;
            }
        }
    }

    print "]\n";
    if ($depth) { print STDERR "syntax error\n"; }
}
```

```
print STDERR "Done...\n";
```

E.14 susanne_parsing_no_traced_to_rules.prl

```
#!/usr/bin/perl
# susanne_parsing_notrace_to_rules.prl
#
# We assume <spnt> to be a file in the syntax of Susanne parsing with no traces.
# This program produces the corresponding list of rules.
#
# Example of <spnt> (input):
#
#    [ O [ S [ Nns:s [ AT The AT ] [ Nns [ NP1s Fulton NP1s ] [ NNL1cb County
#    NNL1cb ] Nns ] [ JJ Grand JJ ] [ NN1c Jury NN1c ] Nns:s ] [ Vd [ VVDv
#    said VVDv ] Vd ] [ Nns:t [ NPD1 Friday NPD1 ] Nns:t ] [ Fn:o [ Ns:s
#    [ AT1 an AT1 ] [ NN1n investigation NN1n ] [ Po [ IO of IO ] [ Ns [ G
#    [ Nns [ NP1t Atlanta NP1t ] Nns ] [ GG <apos>s GG ] G ] [ JJ recent JJ ]
#    [ JJ primary JJ ] [ NN1n election NN1n ] Ns ] Po ] Ns:s ] [ Vd [ VVDv
#    produced VVDv ] Vd ] [ YIL <ldquo> YIL ] [ Ns:o [ ATn no ATn ] [ NN1u
#    evidence NN1u ] [ YIR <rdquo> YIR ] [ Fn [ CST that CST ] [ Np:s [ DDy
#    any DDy ] [ NN2 irregularities NN2 ] Np:s ] [ Vd [ VVDv took VVDv ] Vd ]
#    [ Ns:o [ NNL1c place NNL1c ] Ns:o ] Fn ] Ns:o ] Fn:o ] S ] [ YF . YF ] O ]
#
# Example of output:
#
#    Nns -> NP1s NNL1cb
#    Nns:s -> AT Nns JJ NN1c
#    Vd -> VVDv
#    Nns:t -> NPD1
#    Nns -> NP1t
#    G -> Nns GG
#    Ns -> G JJ JJ NN1n
#    Po -> IO Ns
#    Ns:s -> AT1 NN1n Po
#    Vd -> VVDv
#    Np:s -> DDy NN2
#    Vd -> VVDv
#    Ns:o -> NNL1c
#    Fn -> CST Np:s Vd Ns:o
#    Ns:o -> ATn NN1u YIR Fn
#    Fn:o -> Ns:s Vd YIL Ns:o
#    S -> Nns:s Vd Nns:t Fn:o
#    O -> S YF
#
# Usage:
#    cat <spnt> | susanne_parsing_no_traced_to_rules.prl

while ($line = <STDIN>) {
    chomp $line;
    @data = split(" ", $line);
    @rules_stack = ();
    $depth = 0;

    $i = 0;
    while ($i <= $#data) {
        SWITCH: {
            if ($data[$i] eq "[") {
```

```
                # treatment for sequence "[ NT"
                if (($data[$i+1] eq "YPL") or ($data[$i+1] eq "YPR")) {
                    # special treatment for sequences "[ YPL [ YPL ]" and "[ YPR ] YPR ]"
                    $rules_stack[$#rules_stack] .= " $data[$i+1]";
                    $i += 5;
                } else {
                    # normal processing
                    $depth++;
                    if ($#rules_stack < 0) {
                        $rules_stack[0] = "$data[$i+1] ->";
                    } else {
                        $rules_stack[$#rules_stack] .= " $data[$i+1]";
                        push (@rules_stack, "$data[$i+1] ->");
                    }
                    $i += 2;
                }
                last SWITCH;
            }

            if ($data[$i] eq "]") {
                $depth--;
                print "$rules_stack[$#rules_stack]\n";
                pop (@rules_stack);
                $i++;
                last SWITCH;
            }

            if ($data[$i+1] eq "]") {
                # treatment for sequences "NT ]", where NT is the same non-terminal
                # as the one in the left hand side of the rule in the top of the
                # rules stack
                $i++;
            } else {
                # treatment for sequences "word NT ]", where NT is the tag of the word
                if ($data[$i+1] ne "YG") {
                    $depth--;
                    pop (@rules_stack);
                    $i += 3;
                } else {
                    print STDERR "error: this parsing contains traces\n";
                    exit;
                }
            }
        }
    }

    if (($depth) or ($#rules_stack >= 0)) { print STDERR "syntax error\n"; }
}
```

E.15 susanne_grammar_to_slptoolkit_grammar.prl

```
#!/usr/bin/perl
# susanne_grammar_to_slptoolkit_grammar.prl
#
# We assume <tag-table> to be a file with lines
#
#    1:tag1
```

```
#    2:tag2
#    ...
#    n:tagn
#
# and <grammar> to be a files with rules
#
#    NT1 -> NT2 NT3 ... NTn (probability)
#
# This program translates <grammar> into a grammar in SLPToolKit syntax.
# Basically, it replaces:
#
#    tagi with :i
#
# and for the rest of non-terminals, it replaces:
#
#    : with _
#    - with $
#    " with ^
#
# Example: (we assume line "350:RRR" belongs to <tag-table>)
#
#    old rule:   R:t -> RRR Ns Fc (0.00564971)
#    new rule:   R_t -> :350 Ns Fc (0.00564971)
#
# Usage:
#    susanne_grammar_to_slptoolkit_grammar.prl <tag-table> <grammar>

($#ARGV + 1 == 2) or die "\nUsage: $0 <tag-table> <grammar>\n\n";
open (TT, "$ARGV[0]") or die "Can't open $ARGV[0]: $!\n";
open (SG, "$ARGV[1]") or die "Can't open $ARGV[1]: $!\n";

print STDERR "Loading tag table...\n";
while ($lineTT = <TT>) {
    chomp $lineTT;
    ($n, $t) = split (":", $lineTT);
    $tag_table{$t} = $n;
}

print STDERR "Calculating new rules...\n";
while ($lineSG = <SG>) {
    $lineSG =~ s/:/_/g;
    $lineSG =~ s/-/\$/g;
    $lineSG =~ s/\"/\^/g;
    @symbols = split (" ", $lineSG);
    print "$symbols[0] -> ";

    $before_last = $#symbols - 1;
    foreach $s (@symbols[2..$before_last]) {
        $ns = $tag_table{$s};
        if ($ns ne "") {
            print ":$ns ";
        } else {
            print "$s ";
        }
    }

    print "$symbols[$#symbols]\n";
}
```

```
print STDERR "Done...\n";
```

E.16 susanne_parsing_to_tagged_corpus.prl

```
#!/usr/bin/perl
# susanne_parsing_to_tagged_corpus.prl
#
# We assume <sp> to be a file in the syntax of Susanne parsing.
# This program produces the corresponding tagged corpus.
#
# Example of input:
#
#    [ O [ S [ Nns:s [ AT The AT ] [ Nns [ NP1s Fulton NP1s ] [ NNL1cb County
#    NNL1cb ] Nns ] [ JJ Grand JJ ] [ NN1c Jury NN1c ] Nns:s ] [ Vd [ VVDv
#    said VVDv ] Vd ] [ Nns:t [ NPD1 Friday NPD1 ] Nns:t ] [ Fn:o [ Ns:s
#    [ AT1 an AT1 ] [ NN1n investigation NN1n ] [ Po [ IO of IO ] [ Ns [ G
#    [ Nns [ NP1t Atlanta NP1t ] Nns ] [ GG <apos>s GG ] G ] [ JJ recent JJ ]
#    [ JJ primary JJ ] [ NN1n election NN1n ] Ns ] Po ] Ns:s ] [ Vd [ VVDv
#    produced VVDv ] Vd ] [ YIL <ldquo> YIL ] [ Ns:o [ ATn no ATn ] [ NN1u
#    evidence NN1u ] [ YIR <rdquo> YIR ] [ Fn [ CST that CST ] [ Np:s [ DDy
#    any DDy ] [ NN2 irregularities NN2 ] Np:s ] [ Vd [ VVDv took VVDv ] Vd ]
#    [ Ns:o [ NNL1c place NNL1c ] Ns:o ] Fn ] Ns:o ] Fn:o ] S ] [ YF . YF ] O ]
#
# Example of output:
#
#    The/AT Fulton/NP1s County/NNL1cb Grand/JJ Jury/NN1c said/VVDv Friday/NPD1
#    an/AT1 investigation/NN1n of/IO Atlanta/NP1t <apos>s/GG recent/JJ primary/JJ
#    election/NN1n produced/VVDv <ldquo>/YIL no/ATn evidence/NN1u <rdquo>/YIR
#    that/CST any/DDy irregularities/NN2 took/VVDv place/NNL1c ./YF
#
# Usage:
#    cat <sp> | susanne_parsing_to_tagged_corpus.prl

while ($line = <STDIN>) {
    chomp $line;
    @data = split(" ", $line);
    $first_word = 1;
    $depth = 0;

    $i = 0;
    while ($i <= $#data) {
        SWITCH: {
            if ($data[$i] eq "[") {
                # treatment for sequence "[ NT"
                if (($data[$i+1] eq "YPL") or ($data[$i+1] eq "YPR")) {
                    # special treatment for sequences "[ YPL [ YPL ]" and "[ YPR ] YPR ]"
                    if ($first_word) { $first_word = 0; } else { print " "; }
                    print "$data[$i+2]/$data[$i+1]";
                    $i += 5;
                } else {
                    # normal processing
                    $depth++;
                    $i += 2;
                }
                last SWITCH;
            }
```

```
            if ($data[$i] eq "]") {
                $depth--;
                $i++;
                last SWITCH;
            }

            if ($data[$i+1] eq "]") {
                # treatment for sequences "NT ]", where NT is the same non-terminal
                # as the one in the left hand side of the rule in the top of the
                # rules stack
                $i++;
            } else {
                # treatment for sequences "word NT ]", where NT is the tag of the word
                $depth--;
                if ($data[$i+1] ne "YG") { # YG means traced parsing
                    if ($first_word) { $first_word = 0; } else { print " "; }
                    print "$data[$i]/$data[$i+1]";
                }
                $i += 3;
            }
        }
    }

    print "\n";
    if ($depth) { print STDERR "syntax error\n"; }
}
```

E.17 tagged_corpus_to_untagged_corpus.prl

```
#!/usr/bin/perl
# tagged_corpus_to_untagged_corpus.prl
#
# We assume <corpus> to be a corpus with lines
#
#    form1/tag1 form2/tag2 ... formn/tagn ./tag.
#
#    (lines could also end with ?/tag? or .../tag... or another
#     punctuation mark, and thus each line is a sentence)
#
# This program uses <corpus> in order to generate another
# corpus with lines
#
#    form1 form2 ... formn .
#
# Usage:
#    cat <corpus> | tagged_corpus_to_untagged_corpus.prl

while ($line = <STDIN>) {
    @words = split(" ", $line);

    $beforelast = $#words - 1;
    foreach $w (@words[0..$beforelast]) {
        ($form, $tag) = split("/", $w);
        print "$form ";
    }
```

```
    ($form, $tag) = split("/", $words[$#words]);
    print "$form\n";
}
```

E.18 test_slptoolkit_on_notrace.prl

```
#!/usr/bin/perl
# test_slptoolkit_on_notrace.prl
#
# This program takes SUSANNE_notrace_PARSING.slptoolkit as a
# reference corpus, parses all the sentences in
# SUSANNE_notrace_UNTAGGED_CORPUS, and generates the file
# results/results_on_notrace which contains one line per sentence with
# the following format:
#
#    <line_number> <number_of_ambiguities> <rank_of_the_reference_parsing>
#
# Example of output:
#    1       530     1
#    2       80      1
#    3       480     1
#    4       5       1
#    5       1       1
#    6       1       1
#    7       2       1
#    8       2       2
#    9       20      1
#    10      5       1
#
# Usage:
#    test_slptoolkit_on_notrace.prl SUSANNE_notrace_PARSING.slptoolkit
#        SUSANNE_notrace_UNTAGGED_CORPUS results/results_on_notrace

($#ARGV + 1 == 3) or die "\nUsage: $0 <reference> <input> <output>\n\n";
open (REF, "$ARGV[0]") or die "Can't open $ARGV[0]: $!\n";
open (INPUT, "$ARGV[1]") or die "Can't open $ARGV[1]: $!\n";
open (OUTPUT, ">$ARGV[2]") or die "Can't open $ARGV[2]: $!\n";

sub numeric_sort { $b <=> $a }

$SUSANNE_DIR = "/users/grana/TreeBank";

$n = 0;
while ($line_INPUT = <INPUT>) {
    $line_REF = <REF>;
    chomp $line_REF;
    $n++;
    open (TEMP_IN, ">temp_in") or die "Can't open temp_in: $!\n";
    print TEMP_IN "$line_INPUT";
    close (TEMP_IN);

    $pid = open (TEMP_OUT,
        "anagram -P -K -i p -s -cg $SUSANNE_DIR/data/lexicon $SUSANNE_DIR/data/grammar-bin
        < temp_in|");

    $p = 0;
    $killed = 0;
```

```
    @t = ();
    $match = 0;
    while (($line_TEMP_OUT = <TEMP_OUT>) && (!$killed)) {
        $p++;
        chomp $line_TEMP_OUT;
        @data = split (" ", $line_TEMP_OUT);

        $probability = $data[$#data];
        $probability =~ s/\(//g;
        $probability =~ s/\)//g;
        push (@t, $probability);

        pop @data;
        $line_TEMP_OUT = join (" ", @data);
        if ($line_TEMP_OUT eq $line_REF) { $match = $probability; }
        if ($p >= 5000) {
            `kill -9 $pid`; $killed = 1;
            open (TEMP_OUT_2,
                "anagram -P -K -s -cg $SUSANNE_DIR/data/lexicon $SUSANNE_DIR/data/grammar-bin
                < temp_in|head|");
            $line_TEMP_OUT_2 = <TEMP_OUT_2>;
            chomp $line_TEMP_OUT_2;
            close (TEMP_OUT_2);
        }
    }
    close (TEMP_OUT);

    if ($killed) {
        print OUTPUT "$n\t$line_TEMP_OUT_2\t0\n";
        print STDERR "$n\t$line_TEMP_OUT_2\t0\n";
    } else {
        if ($match == 0) {
            $m = 0;
        } else {
            @t = sort numeric_sort @t;
            $i = 0;
            while ($t[$i] != $match) { $i++; }
            $m = $i+1;
        }

        print OUTPUT "$n\t$p\t$m\n";
        print STDERR "$n\t$p\t$m\n";
    }

    $command = "rm -f temp_in temp_out";
    `$command`;
}

close (REF);
close (INPUT);
close (OUTPUT);
```

E.19 tagged_corpus_to_sequences_of_tags.prl

```
#!/usr/bin/perl
# tagged_corpus_to_sequences_of_tags.prl
#
```

```
# We assume <tag-table> to be a file with lines
#
#    1:tag1
#    2:tag2
#    ...
#    n:tagn
#
# and <corpus> to be a corpus with lines
#
#    form1/tag1 form2/tag2 ... formn/tagn ./tag.
#
#    (lines could also end with ?/tag? or .../tag... or another
#     punctuation mark, and thus each line is a sentence)
#
# This program uses <corpus> in order to generate another corpus
# without the forms and replaces the tags by the corresponding
# numbers of the tags in the tag table.
#
# Usage:
#    tagged_corpus_to_sequences_of_tags.prl <tag-table> <corpus>

($#ARGV + 1 == 2) or die "\nUsage: $0 <tag-table> <corpus>\n\n";
open (TT, "$ARGV[0]") or die "Can't open $ARGV[0]: $!\n";
open (CP, "$ARGV[1]") or die "Can't open $ARGV[1]: $!\n";

print STDERR "Loading tag table...\n";
while ($lineTT = <TT>) {
    chomp $lineTT;
    ($n, $t) = split (":", $lineTT);
    $tag_table{$t} = $n;
}

print STDERR "Calculating tags...\n";
while ($line = <CP>) {
    @words = split(" ", $line);

    @tags = ();
    foreach $w (@words[0..$#words]) {
        ($form, $tag) = split("/", $w);
        $nt = $tag_table{$tag};
        if ($nt eq "") {
            print STDERR "error: $tag is not in tag-table $ARGV[0]\n";
            exit;
        } else {
            push (@tags, $nt);
        }
    }

    print "@tags\n";
}

print STDERR "Done...\n";
close(TT);
close(CP);
```

E.20 test_slptoolkit_on_notrace_tags.prl

```
#!/usr/bin/perl
# test_slptoolkit_on_notrace_tags.prl
#
# This program takes SUSANNE_notrace_PARSING.tags as a reference
# corpus, parses all the sentences in SUSANNE_notrace_UNTAGGED_CORPUS,
# and generates the file results/results_on_notrace_tags which
# contains one line per sentence with the following format:
#
#    <line_number> <number_of_words> <number_of_errors>
#
# Example of output:
#    1       27      0
#    2       31      0
#    3       36      0
#    4       11      0
#    5       10      0
#    6       16      0
#    7       13      0
#    8       19      0
#    9       24      0
#    10      12      0
#    11      5       0
#    12      6       0
#    13      8       1
#    14      25      0
#    15      2       0
#
# Usage:
#    test_slptoolkit_on_notrace_tags.prl SUSANNE_notrace_PARSING.tags
#        SUSANNE_notrace_UNTAGGED_CORPUS results/results_on_notrace_tags

($#ARGV + 1 == 3) or die "\nUsage: $0 <reference> <input> <output>\n\n";
open (REF, "$ARGV[0]") or die "Can't open $ARGV[0]: $!\n";
open (INPUT, "$ARGV[1]") or die "Can't open $ARGV[1]: $!\n";
open (OUTPUT, ">$ARGV[2]") or die "Can't open $ARGV[2]: $!\n";

$SUSANNE_DIR = "/users/grana/TreeBank";

$n = 0;
while ($line_INPUT = <INPUT>) {
    $line_REF = <REF>;
    chomp $line_REF;
    @ref = split (" ", $line_REF);
    $n++;
    open (TEMP_IN, ">temp_in") or die "Can't open temp_in: $!\n";
    print TEMP_IN "$line_INPUT";
    close (TEMP_IN);

    $pid = open (TEMP_OUT,
        "anagram -K -P -i p -s -cg $SUSANNE_DIR/data/lexicon $SUSANNE_DIR/data/grammar-bin
        < temp_in|");

    $p = 0;
    $best = 0;
    $killed = 0;
    @tags = ();
```

```
while (($line_TEMP_OUT = <TEMP_OUT>) && (!$killed)) {
    $p++;
    chomp $line_TEMP_OUT;
    @data = split (" ", $line_TEMP_OUT);

    $probability = $data[$#data];
    $probability =~ s/\(//g;
    $probability =~ s/\)//g;
    pop @data;

    if ($probability > $best) {
        $best = $probability;
        @tags = ();
        foreach $e (@data) {
            if ($e =~ /^:[0-9]+/) {
                $e =~ s/://g;
                push (@tags, $e);
            }
        }
    }

    if ($p >= 5000) {
        `kill -9 $pid`; $killed = 1;
        open (TEMP_OUT_2,
            "anagram -K -P -s -cg $SUSANNE_DIR/data/lexicon $SUSANNE_DIR/data/grammar-bin
            < temp_in|head|");
        $line_TEMP_OUT_2 = <TEMP_OUT_2>;
        chomp $line_TEMP_OUT_2;
        close (TEMP_OUT_2);
    }
}
close (TEMP_OUT);

if ($#tags != $#ref) {
    print OUTPUT "error in line $n: different lengths\t";
    print STDERR "error in line $n: different lengths\t";
}

$errors = 0;
foreach $w (0..$#tags) {
    if ($tags[$w] ne $ref[$w]) {
        $errors++;
    }
}
$words = $#tags + 1;

print OUTPUT "$n\t$words\t$errors";
print STDERR "$n\t$words\t$errors";

if ($killed) {
    print OUTPUT "\t+5000\t$line_TEMP_OUT_2";
    print STDERR "\t+5000\t$line_TEMP_OUT_2";
}

print OUTPUT "\n";
print STDERR "\n";

$command = "rm -f temp_in temp_out";
```

```
    `$command`;
}

close (REF);
close (INPUT);
close (OUTPUT);
```

E.21 test_slptoolkit_on_trace.prl

```
#!/usr/bin/perl
# test_slptoolkit_on_trace.prl
#
# This program parses all the sentences in
# SUSANNE_trace_UNTAGGED_CORPUS, and generates the file
# results/results_on_trace which contains one line per sentence with
# the following format:
#
#    <line_number> <number_of_ambiguities>
#
# Example of output:
#    1       0
#    2       0
#    3       0
#    4       0
#    5       0
#    6       0
#    7       0
#    8       0
#    9       0
#    10      0
#
# Usage:
#    test_slptoolkit_on_trace.prl SUSANNE_trace_UNTAGGED_CORPUS results_on_trace

($#ARGV + 1 == 2) or die "\nUsage: $0 <input> <output>\n\n";
open (INPUT, "$ARGV[0]") or die "Can't open $ARGV[0]: $!\n";
open (OUTPUT, ">$ARGV[1]") or die "Can't open $ARGV[1]: $!\n";

$SUSANNE_DIR = "/users/grana/TreeBank";

$n = 0;
while ($line_INPUT = <INPUT>) {
    $n++;
    open (TMP_IN, ">tmp_in") or die "Can't open tmp_in: $!\n";
    print TMP_IN "$line_INPUT";
    close (TMP_IN);

    $pid = open (TMP_OUT,
        "anagram -P -K -i p -s -cg $SUSANNE_DIR/data/lexicon $SUSANNE_DIR/data/grammar-bin
        < tmp_in|");

    $p = 0;
    $killed = 0;
    while (($line_TMP_OUT = <TMP_OUT>) && (!$killed)) {
        $p++;
        if ($p >= 5000) {
            `kill -9 $pid`; $killed = 1;
```

```
            open (TMP_OUT_2,
                "anagram -P -K -s -cg $SUSANNE_DIR/data/lexicon $SUSANNE_DIR/data/grammar-bin
                < tmp_in|head|");
            $line_TMP_OUT_2 = <TMP_OUT_2>;
            chomp $line_TMP_OUT_2;
            close (TMP_OUT_2);
        }
    }
    close (TMP_OUT);

    if ($killed) {
        print OUTPUT "$n\t$line_TMP_OUT_2\n";
        print STDERR "$n\t$line_TMP_OUT_2\n";
    } else {
        print OUTPUT "$n\t$p\n";
        print STDERR "$n\t$p\n";
    }

    $command = "rm -f tmp_in tmp_out";
    `$command`;
}

close (INPUT);
close (OUTPUT);
```

E.22 remove_lines_by_number.prl

```
#!/usr/bin/perl
# remove_lines_by_number.prl
#
# This program removes from <source-file> the lines
# in the positions contained in <list-of-numbers>.
#
# Usage:
#    remove_lines_by_number.prl <list-of-numbers> <source-file>

($#ARGV + 1 == 2) or die "\nUsage: $0 <list-of-numbers> <source-file>\n\n";
open (NUMBERS, "$ARGV[0]") or die "Can't open $ARGV[0]: $!\n";
open (SOURCE, "$ARGV[1]") or die "Can't open $ARGV[1]: $!\n";

while ($line = <NUMBERS>) {
    chomp $line;
    $numbers{$line} = 1;
}

$i = 1;
while ($line = <SOURCE>) {
    if ($numbers{$i} eq "") { print "$line"; }
    $i++;
}

close (NUMBERS);
close (SOURCE);
```

E.23 divide_in_two_by_number_of_words.prl

```
#!/usr/bin/perl
# divide_in_two_by_number_of_words.prl
#
# This program splits <source-file> in two parts:
#    - <file-less-equal> containing the sentences with
#      <number-of-words> words or less
#    - <file-greater-than> containing the sentences with
#      more than <number-of-words> words
#
# Usage:
#    divide_in_two_by_number_of_words.prl
#       <number-of-words> <source-file> <file-less-equal> <file-greater-than>

($#ARGV + 1 == 4) or
    die "\nUsage: $0 <number-of-words> <source-file> <file-less-equal> <file-greater-than>\n\n";
($ARGV[0] > 0) or die "$ARGV[0] is not a valid number\n";
open (SOURCE,  "$ARGV[1]") or die "Can't open $ARGV[1]: $!\n";
open (FILE_LE, ">$ARGV[2]") or die "Can't open $ARGV[2]: $!\n";
open (FILE_GT, ">$ARGV[3]") or die "Can't open $ARGV[3]: $!\n";

while ($line = <SOURCE>) {
    @data = split(" ", $line);
    if ($#data < $ARGV[0]) {
        print FILE_LE "$line";
    } else {
        print FILE_GT "$line";
    }
}

close(SOURCE);
close(FILE_LE);
close(FILE_GT);
```

E.24 forest.c

`forest.c` is not a PERL script, but a C program based on `anagram.c` that directly uses the resources implemented in the SLPTOOLKIT library of functions. The following piece of code is located inside the function `analyse`, and is the main different between `forest.c` and `anagram.c`:

```
...
if (iteratif) {
  /* begin grana */
  if (forest) {
    for (i = 1; i <= resultat->table->taille; i++) {
      for (j = 1; j <= resultat->table->taille - i + 1; j++) {
        rez.contenu = (Brique_cyk *) 0;
        if (resultat->table != (Table_CYK *) 0) {
         do {
           suite =
             Parcours_Cyk_Iteratif(resultat->table, &meta,
                                   i, j, toto, &pile,
                                   (int (*)(const char *, ... )) 0, FAUX, FAUX,
                                   &proba);
           if (rez.nombre != 0) {
             subt_aux = subt;
             for (k = 0; k < rez.nombre; k++) {
```

```
                sprintf(subt_aux, ":" FMT_morpho " ", rez.contenu[k].cms);
                while (*subt_aux) subt_aux++;
                monfree((void **) &(rez.contenu[k].mot));
              }
              INSERT_STRING_LIST (&subtrees, subt, proba);
            }
          } while (suite == VRAI); }
         while (!IS_EMPTY_STRING_LIST (subtrees)) {
          printf ("%d %d %d %d (%f) %s\n", line, i, j,
                  CAR_STRING_LIST_counter (subtrees),
                  CAR_STRING_LIST_proba (subtrees),
                  CAR_STRING_LIST_string (subtrees));
          NEXTL_STRING_LIST (&subtrees);
        }
        Libere_Briques_Cyk(&rez);
      }
    }
  } else
  /* end grana */
...
```

E.25 combine_forest_reference.prl

```
#!/usr/bin/perl
# combine_forest_reference.prl
#
# This program takes <forest> and, for each sentence, inserts directly
# below all the subtrees of a given sentence the corresponding
# sequence of tags taken from <reference> for that sentence.
#
# Usage:
#    combine_forest_reference.prl <forest> <reference>

($#ARGV + 1 == 2) or die "\nUsage: $0 <forest> <reference>\n\n";
open (FOREST, "$ARGV[0]") or die "Can't open $ARGV[0]: $!\n";
open (REF, "$ARGV[1]") or die "Can't open $ARGV[1]: $!\n";

$n = 1;
while ($line = <FOREST>) {
    @data = split(" ", $line);
    $line_number = shift(@data);
    if ($line_number == $n) {
        print "$line_number @data\n";
    } else {
        $line_ref = <REF>;
        @data_ref = split(" ", $line_ref);
        print "Ref ";
        $t = $#data_ref - 1;
        foreach $i (0..$t) {
            print ":$data_ref[$i] ";
        }
        print ":$data_ref[$#data_ref]\n";
        print "$line_number @data\n";
        $n++;
    }
}
```

```
$line_ref = <REF>;
@data_ref = split(" ", $line_ref);
print "Ref ";
$t = $#data_ref - 1;
foreach $i (0..$t) {
    print ":$data_ref[$i] ";
}
print ":$data_ref[$#data_ref]\n";

close(FOREST);
close(REF);
```

E.26 strategy_1.prl

```
#!/usr/bin/perl
# strategy_1.prl
#
# This program uses a forest to recursively build a sequence
# of tags. The criterion is to always take the longest and
# most probable tree.
#
# Output format for each line is:
#    #F   #AF-
#
# Usage:
#    cat SUSANNE_trace_np_25_FOREST | strategy_1.prl > results/results_stgy_1_trace_np_25
#    cat SUSANNE_trace_np_50_FOREST | strategy_1.prl > results/results_stgy_1_trace_np_50

sub most_probable_tree {
    my ($mpt_length, $mpt_start) = @_;
    my ($i, $j, $k, $ri, $rj, $rk, $found, $maxj, $cardinal, $mp, $np);

    $i = $mpt_length;
    $found = 0;
    while (($i > 0) && (!$found)){
        $maxj = $mpt_length + $mpt_start - $i;
        foreach $j ($mpt_start..$maxj) {
            $cardinal = $cyk_cardinal{"$i,$j"};
            if ($cardinal ne "") {
                $ri = $i;
                foreach $k (1..$cardinal) {
                    if (!$found) {
                        $found++;
                        $rj = $j;
                        $rk = $k;
                        $mp = $cyk_proba{"$i,$j,$k"};
                    } else {
                        $np = $cyk_proba{"$i,$j,$k"};
                        if ($np > $mp) { $mp = $np; $rj = $j; $rk = $k; }
                    }
                }
            }
        }
        $i--;
    }
    return ($ri, $rj, $rk);
}
```

```
sub fill_table {
    my ($ft_length, $ft_start) = @_;
    my ($tags_left, $tags_right);

    my ($i, $j, $k) = most_probable_tree ($ft_length, $ft_start);
    my $tags = $cyk_tags{"$i,$j,$k"};

    if (($i == $ft_length) && ($j == $ft_start)) { return $tags; }

    if ($j == $ft_start) {
        $tags_right = fill_table ($ft_length - $i, $i + $j);
        return "$tags $tags_right";
    }

    if ($i + $j - 1 == $ft_length + $ft_start - 1) {
        $tags_left = fill_table ($ft_length - $i, $ft_start);
        return "$tags_left $tags";
    }

    $tags_left = fill_table ($j - $ft_start, $ft_start);
    $tags_right = fill_table ($ft_length + $ft_start - $i - $j, $i + $j);
    return "$tags_left $tags $tags_right";

}

$old_length = 0;
$old_start = 0;
$ntrees = 0;

$af_errors = 0;

while ($line = <STDIN>) {
    chomp $line;
    @data = split (" ", $line);
    $line_id = shift (@data);
    if ($line_id ne "Ref") {

        ### Load forest #####

        $length  = shift (@data);
        $start   = shift (@data);
        $counter = shift (@data);
        $proba   = shift (@data); $proba =~ s/\(//g; $proba =~ s/\)//g;

        if ($length == $old_length) {
            if ($start == $old_start) {
                $ntrees++;
            } else {
                $cyk_cardinal{"$length,$old_start"} = $ntrees;
                $old_start = $start;
                $ntrees = 1;
            }
        } else {
            $cyk_cardinal{"$old_length,$old_start"} = $ntrees;
            $old_length = $length;
            $old_start = $start;
            $ntrees = 1;
```

```
        }

        $cyk_counter{"$length,$start,$ntrees"} = $counter;
        $cyk_proba{"$length,$start,$ntrees"} = $proba;
        $cyk_tags{"$length,$start,$ntrees"} = join (" ", @data);

    } else {

        $cyk_cardinal{"$old_length,$old_start"} = $ntrees;
        $old_length = 0;
        $old_start = 0;
        $ntrees = 0;

        $length = $#data + 1;
        $ref = join (" ", @data);

        ### Begin strategy ########################################################

        $seq = fill_table ($length, 1);
        @seq = split (" ", $seq);

        foreach $i (0..$#data) {
            if ($data[$i] ne $seq[$i]) { $af_errors++; }
        }
        print "$length\t$af_errors\n";

        $af_errors = 0;

        ### End strategy ##########################################################

        %cyk_cardinal = "";
        %cyk_counter = "";
        %cyk_proba = "";
        %cyk_tags = "";

    }
}
```

E.27 strategy_2.prl

```
#!/usr/bin/perl
# strategy_2.prl
#
# This program uses a forest to recursively build several sequences
# of tags. The criterion is to always take the longest and
# most probable tree, but instead of starting at the top of the CYK
# table, we apply the same procedure to each cell. Therefore, the
# lowest number of sequences is 1, and the highest is (n * (n+1)) / 2,
# where n is the number of words in the sentence.
#
# Output format for each line is:
#    #F   #AF- (the lowest value obtained when we compare with the reference)
#
# Usage:
#    cat SUSANNE_trace_np_25_FOREST | strategy_2.prl > results/results_stgy_2_trace_np_25
#    cat SUSANNE_trace_np_50_FOREST | strategy_2.prl > results/results_stgy_2_trace_np_50
```

```
sub most_probable_tree {
    my ($mpt_length, $mpt_start) = @_;
    my ($i, $j, $k, $ri, $rj, $rk, $found, $maxj, $cardinal, $mp, $np);

    $i = $mpt_length;
    $found = 0;
    while (($i > 0) && (!$found)){
        $maxj = $mpt_length + $mpt_start - $i;
        foreach $j ($mpt_start..$maxj) {
            $cardinal = $cyk_cardinal{"$i,$j"};
            if ($cardinal ne "") {
                $ri = $i;
                foreach $k (1..$cardinal) {
                    if (!$found) {
                        $found++;
                        $rj = $j;
                        $rk = $k;
                        $mp = $cyk_proba{"$i,$j,$k"};
                    } else {
                        $np = $cyk_proba{"$i,$j,$k"};
                        if ($np > $mp) { $mp = $np; $rj = $j; $rk = $k; }
                    }
                }
            }
        }
        $i--;
    }
    return ($ri, $rj, $rk);
}

sub fill_table {
    my ($ft_length, $ft_start) = @_;
    my ($tags_left, $tags_right);

    my ($i, $j, $k) = most_probable_tree ($ft_length, $ft_start);
    my $tags = $cyk_tags{"$i,$j,$k"};

    if (($i == $ft_length) && ($j == $ft_start)) { return $tags; }

    if ($j == $ft_start) {
        $tags_right = fill_table ($ft_length - $i, $i + $j);
        return "$tags $tags_right";
    }

    if ($i + $j - 1 == $ft_length + $ft_start - 1) {
        $tags_left = fill_table ($ft_length - $i, $ft_start);
        return "$tags_left $tags";
    }

    $tags_left = fill_table ($j - $ft_start, $ft_start);
    $tags_right = fill_table ($ft_length + $ft_start - $i - $j, $i + $j);
    return "$tags_left $tags $tags_right";

}

$old_length = 0;
$old_start = 0;
$ntrees = 0;
```

```
$af_errors = 0;

while ($line = <STDIN>) {
    chomp $line;
    @data = split (" ", $line);
    $line_id = shift (@data);
    if ($line_id ne "Ref") {

        ### Load forest #####

        $length  = shift (@data);
        $start   = shift (@data);
        $counter = shift (@data);
        $proba   = shift (@data); $proba =~ s/\(//g; $proba =~ s/\)//g;

        if ($length == $old_length) {
            if ($start == $old_start) {
                $ntrees++;
            } else {
                $cyk_cardinal{"$length,$old_start"} = $ntrees;
                $old_start = $start;
                $ntrees = 1;
            }
        } else {
            $cyk_cardinal{"$old_length,$old_start"} = $ntrees;
            $old_length = $length;
            $old_start = $start;
            $ntrees = 1;
        }

        $cyk_counter{"$length,$start,$ntrees"} = $counter;
        $cyk_proba{"$length,$start,$ntrees"} = $proba;
        $cyk_tags{"$length,$start,$ntrees"} = join (" ", @data);

    } else {

        $cyk_cardinal{"$old_length,$old_start"} = $ntrees;
        $old_length = 0;
        $old_start = 0;
        $ntrees = 0;

        $length = $#data + 1;
        $ref = join (" ", @data);

        ### Begin strategy ###########################################################

        $min_af_errors = $length;

        foreach $i (1..$length) {
            $maxj = $length - $i + 1;
            foreach $j (1..$maxj) {
                $maxk = $cyk_cardinal{"$i,$j"};
                if ($maxk ne "") {
                    foreach $k (1..$maxk) {
                        if ($k == 1) {
                            $mpk = $k;
                            $mp = $cyk_proba{"$i,$j,$k"};
```

```
                        } else {
                            $nmp = $cyk_proba{"$i,$j,$k"};
                            if ($nmp > $mp) { $mp = $nmp; $mpk = $k; }
                        }
                    }

                    $tags = $cyk_tags{"$i,$j,$mpk"};

                    if ($i == $length) {
                        $seq = $tags;
                    } else {
                        if ($j == 1) {
                            $tags_right = fill_table ($length - $i, $i + $j);
                            $seq = "$tags $tags_right";
                        } else {
                            if ($i + $j - 1 == $length) {
                                $tags_left = fill_table ($length - $i, 1);
                                $seq = "$tags_left $tags";
                            } else {
                                $tags_left = fill_table ($j - 1, 1);
                                $tags_right = fill_table ($length - $i - $j + 1, $i + $j);
                                $seq = "$tags_left $tags $tags_right";
                            }
                        }
                    }

                    @seq = split (" ", $seq);

                    $af_errors = 0;
                    foreach $i (0..$#data) {
                        if ($data[$i] ne $seq[$i]) { $af_errors++; }
                    }
                    if ($af_errors < $min_af_errors) { $min_af_errors = $af_errors; }
                }
            }
        }

        print "$length\t$min_af_errors\n";

        $af_errors = 0;

        ### End strategy ###########################################################

        %cyk_cardinal = "";
        %cyk_counter = "";
        %cyk_proba = "";
        %cyk_tags = "";

    }
}
```