Does tagging help parsing? A case study on finite state parsing

Atro Voutilainen

Research Unit for Multilingual Language Technology
Department of General Linguistics
P.O. Box 4 (Keskuskatu 8, 7th floor)
FIN-00014 University of Helsinki, Finland
Atro.Voutilainen@ling.helsinki.fi

Abstract. The usefulness of POS taggers for syntactic parsing is a question little addressed in the literature. Because taggers reduce ambiguity from the parser's input, parsing is commonly supposed to become faster, and the result less ambiguous. On the other hand, tagging errors probably reduce the parser's recognition rate, so this drawback may outweigh the possible advantages. This paper empirically investigates these issues using two different rule-based morphological disambiguators as preprocessor of a wide-coverage finite-state parser of English. With these rule-based taggers, the parser's output becomes less ambiguous without a considerable penalty to recognition rate. Parsing speed increases slightly, not decisively.

1 Introduction

1.1 The parsing problem

Most application-oriented natural-language parsers are computer programs that try to automatically assign a syntactic structure onto input sentences, by using a formal language model, e.g. a large grammar. In a sense, a syntactic parser tries to choose from a multitude of imaginable syntactic sentence analyses the alternative that is the most appropriate for the input sentence. So syntactic parsing can be viewed as a massive disambiguation task where the main problems are:

(i) Determining the correct sentence analysis, especially in the case of long sentences, may require a lot of time and memory (computational problem)
(ii) Deciding between possible analyses may be difficult due to (i) the inherent syntactic ambiguity of the sentence or (ii) the shortages in the specificity of the parser's language model (linguistic problem)
(iii) Assigning a syntactic analysis to an acceptable sentence may fail because the parser's language model does not account for the syntactic structures that emerge as the sentence (linguistic problem)

1.2 A possible partial solution

A solution often proposed to these problems is use of a tagger as a front-end of the parser. Now what is a tagger, and how are they supposed to help parsers?

What are taggers? Programs variously known as part-of-speech taggers, POS taggers or morphological taggers typically contain three analytic stages:

- The *tokeniser* identifies words, punctuation marks and sentence boundaries.
- The *lexical analyser* assigns possible analyses to each word. Some words can receive multiple analyses; for instance the sentence *Those girls sing well* can be analysed as follows:

```
Those_DET_P
girls_Npl
sing_Vpres_Vinf_Vimp_Vsbj
well_ADV_N_Vpres_Vinf_Vimp_Vsbj
.
```

  Here *Those* gets two alternative analyses (a determiner and a pronoun analysis), while *well* gets as many as six alternative analyses.
- The *disambiguator* resolves lexical ambiguity by removing analyses that seem superfluous according to the language model used by the program; for instance:

```
Those_DET
girls_Npl
sing_Vpres
well_ADV
.
```

  In this case all words became fully disambiguated, but, to reduce the risk of removing the correct analysis, some of the most difficult ambiguities can optionally be left pending in many tagging programs (so other modules e.g. the parser, can choose between them if necessary).

The design of the disambiguator is the hardest subproblem in tagging, and several solutions have been proposed for improving the reliability of the disambiguator's language model (data-driven models automatically generated from pretagged texts; linguistic models with manually developed disambiguation grammars; hybrid models).

Why are taggers supposed to be useful? As for the supposed usefulness of taggers for parsing, the following assumptions are often made:

(i) The tagger rapidly decreases the ambiguity of the sentence, as a consequence of which the computationally heavier syntactic parser has to deal with less ambiguity. So the computational problem should become smaller and parsing faster.
(ii) The tagger resolves some ambiguities not addressed by the syntactic parser's language model, so the parser is expected to have a better chance to find the correct analysis with the tagger than without it.

At the first flush, these assumptions may seem self-evidently true. However, also sceptical views are possible. One could argue that taggers do not give any real advantage for mature syntactic parsers, e.g. for the following reasons:

(i) taggers resolve mainly local, 'easy' ambiguities that would be resolved by the parser in any case with very little extra computational load, so it is questionable whether a mature syntactic parser would gain anything in terms of speed or accuracy from using a tagger,
(ii) taggers make so many mispredictions that the possible gains in parsing time, or average number of syntactic parses, is more than counteracted by the decrease in the parser's recall: for many sentences no good parse is available if correct POS tags have been lost, and

(iii) even if there were a tagger with a satisfactory recall and precision, making it would take so much effort, either in the form of annotating big training corpora, or writing rules, that the same effort would be more beneficially spent on developing the parser itself.

So the assumed relevance of tagging for parsing seems to be an open, empirical question. What does the literature say on the subject?

1.3 Earlier studies

The computational linguistics literature seems to contain very few evaluations about using taggers in parsing. Three studies are examined next.

- A well-known paper on using statistical taggers in statistical parsing is one by Charniak et al. [1]. In their experiments, they use two kinds of statistical tagger: the *single tagger* outputs fully disambiguated text, while the *multiple tagger* leaves the hardest ambiguities unresolved to decrease the number of its mispredictions. Contrary to expectations, their experiments suggest that the statistical parser is no better in resolving the hardest morphological ambiguities than the single tagger, so passing the most difficult ambiguities on to the assumedly more informed syntactic language model does not seem practically motivated.
- A paper by Wauschkuhn [17] examines the use of a statistical HMM tagger for German as a front-end of a syntactic parser that uses a hand-coded grammar. The experiments suggest that a tagger reduces the ambiguity of the syntactic parser's output, but only with a considerable penalty in terms of the poorer recognition rate of the parser. The experiments give the impression that taggers, at least statistical ones, are not particularly useful for improving a parser's accuracy.
- Results more favourable for tagging are reported by Oflazer and Kuruöz [7]. They report that a rule-based tagger of Turkish (that uses a human disambiguator as its final component) improved the speed of a LFG parser of Turkish using a non-trivial grammar by a factor of 2.38, while the average number of parses per sentence fell from 5.78 to 3.30. However, they do not report the figures for sentence recognition: how many sentences got a parse with the tagger and how many without. Also their test data was rather small: 80 sentences with an average sentence length of 5.7 words only. Several questions remain open:
  - How much does the *fully automatic* part of the tagger improve performance?
  - What is the behaviour of the systems with longer sentences?
  - How is the parser's recognition rate affected?
  - How much would the tagger benefit a parser with a more mature grammar?

1.4 Structure of this paper

In this paper, we present experiments with two taggers and one parser. The parser is a reductionistic dependency-oriented finite-state parser of English that represents utterances with morphological and syntactic tags. The parser consists of the following components: (i) a tokeniser; (ii) a morphological analyser; (iii) a simple lookup program for introducing all possible syntactic analyses as alternatives for each word and word-boundary; and (iv) a finite-state syntactic parser (actually a syntactic *disambiguator*) that discards those sentence readings that violate the parser's grammar.

This setup contains no specific module for resolving morphological ambiguities that arise in lexical analysis. The syntactic grammar actually suffices for resolving many morphological ambiguities as a side-effect of proper syntactic parsing.

However, a morphological disambiguator can *optionally* be included in the setup, directly after morphological analysis. The disambiguators referred to in this paper are linguistic constraint-based systems. Ambiguities occurring in unspecified contexts are not resolved, so these disambiguators can produce ambiguous output.

In this paper we report experiments where the modular setup uses the following disambiguation modules:

- No disambiguation. Only the finite state syntactic parser is used for ambiguity resolution.
- A small disambiguator whose 149 rules were written during one day. This module discards over 70% of all extra morphological readings.
- A mature disambiguator whose 3,500 rules were written in the course of several years. This module discards about 95% of all extra morphological readings with a minimal error rate.

The data is new to the system, and it consists of three corpora, each with 200 sentences. In the experiments, we consider the following issues:

- *Syntactic ambiguity before finite state disambiguation*: how is the ambiguity rate of the syntactic disambiguator's input reduced by different morphological disambiguators?
- *The parser's recognition rate*: how many sentences does the finite state parser recognise with different morphological disambiguators?
- *Multiple analyses*: how much syntactic ambiguity is produced by the different setups?
- *Parsing time*: how much does the use of different disambiguators affect parsing time?

2 The finite state parser

The finite state parser outlined in this section is described in greater detail in Tapanainen [11] and Voutilainen [14].

2.1 Grammatical representation

Let us describe the syntactic representation with an example. The parser produces the following analysis for the sentence *The man who is fond of singing this aria killed his father* (some morphological information is deleted for readability):

```
                                  @@
the      DET     @>N               @
man      N       @SUBJ             @<
who      PRON    @SUBJ             @
be       V       @MV       N<@     @
fond     A       @SC               @
of       PREP    @N<               @
sing     PCP1    @mv     P<<@      @
this     DET     @>N               @
aria     N       @obj              @>
kill     V       @MV   MAINC@      @
he       PRON    @>N               @
father   N       @OBJ              @
@fullstop                          @@
```

The representation consists of base-forms and various kinds of tags. "@@" indicates sentence boundaries; the centre-embedded finite clause "who is fond of singing this aria" is flanked by the clause boundary tags @< and @> and its function is postmodifying, as indicated with the second tag N<@ of "be", the main verb (@MV) of the clause. The pronoun "who" is the subject (@SUBJ) of this clause, and the adjective "fond" is the subject complement (@SC) that is followed by the postmodifying (@N<) prepositional phrase starting with "of", whose complement is the nonfinite main verb (@mv) "sing" that has the noun "aria" as its object (@obj) (note that the lower case is reserved for functions in nonfinite clauses).

The matrix clause "The man killed his father" is a finite main clause (MAINC@) whose main verb (@MV) is "kill". The subject (@SUBJ) of the finite clause is the noun "man", while the noun "father" is the object in the finite clause (@OBJ). The word "father" has one premodifier (@>N), namely the genitive pronoun "he".

This representation is designed to follow the principle of surface-syntacticity: distinctions not motivated by surface grammatical phenomena, e.g. many attachment and coordination problems, are avoided by making the syntactic representation sufficiently underspecific in the description of grammatically (if not semantically) unresolvable distinctions.

2.2 Analysis routine

The tokeniser identifies words and punctuation marks. The morphological analyser contains a rule-based lexicon and a guesser that assign one or more morphological analyses to each word, cf. the analysis of the word-form "tries"

```
"<tries>"
   "try" <SVO> V PRES SG3 VFIN
   "try" N NOM PL
```

The next step is the introduction of alternative syntactic and word-boundary descriptors with a simple lookup program. After this stage, the sentence "Pete tries." looks like this:

```
(@@ pete <*> <Proper> N NOM SG (:OR @nh @>N @>>P @SUBJ @subj @OBJ @obj
    @IOBJ @iobj @SC @sc @OC @oc @APP @P<< @ADVL)
(:OR @ @/ @< @>) (:OR
 (try <SVO> <SV> <P/for> V PRES SG3 VFIN
    (:OR (@MV MAINC@) (@MV PAREN@) (@MV SUBJ@) (@MV OBJ@) (@MV obj@)
    (@MV IOBJ@) (@MV SC@) (@MV sc@) (@MV OC@) (@MV oc@) (@MV APP@)
    (@MV P<<@) (@MV ADVL@) (@MV N<@)))
 (try N NOM PL (:OR @nh @>N @>>P @SUBJ @subj @OBJ @obj @IOBJ @iobj @SC @sc
    @OC @oc @APP @P<< @ADVL)))
(:OR @ @/ @< @>) @fullstop @@)
```

This compact representation contains $16*4*14*16*4 = 57,344$ different sentence readings. Long sentences easily get $10^{50} - 10^{100}$ different sentence readings at this stage, i.e. the ambiguity problem with this syntactic representation is considerable.

The final stage in this setup is resolution of syntactic ambiguities: those sentence readings that violate even one syntactic rule in the grammar are discarded; the rest are proposed as parses of the sentence.

2.3 Rule formalism

Grammar rules are basically extended regular expressions. A typical rule is the implication rule whereby contextual requirements can be expressed for a distributional (or functional) category. For instance the following partial rule (taken from Voutilainen [12]) about a syntactic form category, namely prepositional phrases,

```
                    PREP =>
                           _ . @ Coord,
                           _ . ..PrepComp,
PassVChain.. <Deferred> . _,
 PostmodCl.. <Deferred> . _,
      WH-Q.. <Deferred> . _;
```

states a number of alternative contexts in which the expression (given left of the arrow) occurs. The underscore shows the position of the expression with regard to the required alternative contexts, expressed as regular expressions. The parser interprets this kind of rule in the following way: whenever a string satisfying the expression left of the arrow is detected, the parser checks whether any of the required contextual expressions are found in the input sentence reading. If a contextual licence is found, the sentence reading is accepted by the rule, otherwise the sentence-reading is rejected.

Another typical rule is the "nowhere" predicate with which the occurrence of a given regular expression can be forbidden. For instance, the predicate **nowhere(VFIN .. VFIN);** forbids the occurrence of two finite verbs in the same finite clause.

These finite-state rules express partial facts about the language, and they are independent of each other in the sense that no particular application order is expected. A sentence reading is accepted by the parser only if it is accepted by each individual rule.

2.4 The grammar

The syntactic grammar contains some 2,600 finite-state rules each of which have been tested and corrected against a manually parsed corpus of about 250,000 words (over 10,000 unambiguously parsed sentences). Each rule in the grammar accepts virtually all parses in this corpus (i.e. a rule may disagree with at most one or two sentences in the corpus, usually when the sentence contains a little-used construction).

The rules are not much restricted by engineering considerations; linguistic truth has been more important. This shows e.g. in the non-locality of many of the rules: the description of many syntactic phenomena seems to require reference to contextual elements in the scope of a finite clause, often even in the scope of the whole sentence, and this kind of globality has been practiced even though this probably results in bigger processing requirements for the finite state disambiguator (many disambiguating decisions have to be delayed e.g. until the end of the clause or sentence, therefore more alternatives have to be kept longer 'alive' than might be the case with very local rules).

Many rules are lexicalised in the sense that some element in the rule is a word (rather than a tag). Though a small purely feature-based grammar may seem more appealing aesthetically or computationally, many useful lexico-grammatical generalisations would be lost if reference to words were not allowed.

To sum up: the finite state disambiguator's task is facilitated by using a reasonably resolvable surface-syntactic grammatical representation, but the parser's task remains computationally rather demanding because of (i) the high initial ambiguity of the input, especially in the case of long sentences, (ii) considerably high number of rules and rule automata, and (iii) the non-locality of the rules. The finite state syntactic disambiguator is clearly faced with a computationally and linguistically very demanding task.

3 Morphological disambiguators

3.1 Mature disambiguator

The mature disambiguator is an early version of a system presently known as EngCG-2 (Samuelsson and Voutilainen [8]). EngCG-2 uses a grammar of 3,500 rules according to the Constraint Grammar framework (Karlsson et al., eds., [4]). The rules are pattern–action statements that, depending on rule type, select a morphological reading as correct (by discarding other readings) or discard a morphological reading as incorrect, when the ambiguity-forming morphological analysis occurs in a context specified with the context-conditions of the constraint. Context-conditions can refer to tags and words in any sentence position; also certain types of word/tag sequences can be used in context-conditions.

An evaluation and comparison of EngCG-2 to a state-of-the-art statistical tagger is reported in (Samuelsson and Voutilainen [8]). In similar circumstances, the error rate of EngCG-2 was an order of magnitude smaller than that of the statistical tagger. On a 266 MHz Pentium running Linux, EngCG-2 tags around 4,000 words per second.[1]

3.2 Small disambiguator

To determine the benefit of using a rule set developed in a short time, one long day was spent on writing a constraint grammar of 149 rules for disambiguating frequent and obviously resolvable ambiguities. As the grammarian's empirical basis, a manually disambiguated benchmark corpus of about 300,000 words was used.

The small grammar was tested against a held-out manually disambiguated (and several times proofread) corpus of 114,388 words with 87,495 superfluous morphological analyses. After the 149 rules were applied to this corpus, there were still 24,458 superfluous analyses, i.e. about 72% of all extra readings were discarded, and the output word contained an average of 1.21 alternative morphological analyses. Of the 63,037 discarded readings, 79 were analysed as contextually legitimate, i.e. of the predictions made by the new tagger, almost 99.9% were correct.

4 Experiments

This section reports the application of the following three setups to new text data:

(i) *Nodis*: the finite state parser is used as such.

[1] Information about testing and licensing the present version of the EngCG-2 tagger is given at the following URL: `http://www.conexor.fi/analysers.html`.

(ii) *Small*: a morphological disambiguation module with 149 rules is used before the finite state parser.
(iii) *Eng*: a morphological disambiguation module with 3,500 rules is used before the finite state parser.

Three text corpora were used as test data:

(i) Data 1: 200 10-word sentences from *The Wall Street Journal*
(ii) Data 2: 200 15-word sentences from *The Wall Street Journal*
(iii) Data 3: 200 20-word sentences from *The Wall Street Journal*

In the word count, punctuation marks were excluded. The data is new to system.

The machine used in the tests is Sun SparcStation 10/30, with 64 MB of RAM.

In the statistics below, the term 'recognition rate' is used. Recognition rate indicates the percentage of sentences that get at least one analysis, correct or incorrect, from the parser. The parser's correctness rate remains to be determined later (but cf. Section 4.2 above).

4.1 Statistics on input ambiguity

Before going to detailed examinations, some statistics on input ambiguity are given. The following table indicates how many readings each word received on an average after possible morphological disambiguation and introduction of syntactic ambiguities. The ambiguity rates are given for morphology and syntax separately.

	Nodis		Small		Eng	
	mor	syn	mor	syn	mor	syn
Data1	1.74	22.81	1.19	16.03	1.04	13.96
Data2	1.78	23.48	1.21	16.70	1.04	14.33
Data3	1.77	23.09	1.23	16.65	1.05	14.13

For instance, after EngCG-2 disambiguation, words in Data 2 received an average 1.04 morphological analyses and 14.33 syntactic analyses.

At the word level, syntactic ambiguity decreases quite considerably even using the small disambiguator, from about 23 syntactic readings per word to some 16.5 syntactic readings per word. Use of the EngCG-2 disambiguator does not contribute much to further decrease of syntactic ambiguity. Overall, syntactic ambiguity at the word level remains quite large, about 14 analyses per word.

However, if we consider the ambiguity rate of the finite state parser's input at the *sentence level* (which is the more common way of looking at ambiguity at the level of syntax), things look more worrying. The following table (next page) presents syntactic ambiguity rates at the sentence level for Data 1 (the 10-word sentences).

When no morphological disambiguation is done, a typical ambiguity rate is 10^{17} sentence readings per input sentence; even after EngCG-2 disambiguation, the typical ambiguity rate is still about 10^{15} sentence readings.

	Number of		sentences
Readings	Nodis	Small	Eng
10^{12}	-	-	4
10^{13}	1	8	9
10^{14}	6	16	27
10^{15}	14	32	48
10^{16}	31	52	49
10^{17}	49	50	33
10^{18}	50	23	17
10^{19}	23	12	7
10^{20}	15	5	2
10^{21}	7	-	2
10^{22}	2	1	-
10^{23}	1	1	-
10^{24}	1	-	-

4.2 Analysis of data 1

All three setups were able to parse Data 1 in the time allowed. Here are some statistics on the different setups:

	Nodis	Small	Eng
0 parses	2% (4)	2.5% (5)	3.5% (7)
1 parses	14.5% (29)	19.5% (39)	37% (74)
1-5 parses	59.5% (119)	64% (128)	80% (160)
1-20 parses	89.5% (179)	91.5% (183)	95.5% (191)
0-10 sec	6% (12)	20.5% (41)	40.5% (81)
0-100 sec	61% (122)	76% (152)	90% (180)

Morphological disambiguation did not considerably affect the recognition rate of the parser. Without morphological disambiguation, the parser gave analyses for 98% of all sentences; the use of the EngCG-2 disambiguator decreased the recognition rate only by 1.5%. Considering that the known strength of the EngCG-2 disambiguator is high recall, the small loss in the number of parses does not seem particularly surprising.

The number of parses decreased even when the small disambiguator was used. The decrease was considerable with EngCG-2, e.g. the rate of sentences receiving 1-5 parses rose from about 60% to 80%. The somewhat unexpected syntactic disambiguating power of the morphological disambiguators is probably due to the lexical nature of the disambiguation grammar (many constraints refer to words, not only to tags). Lexical information has been argued to be an important part of a successful POS tagger (cf. e.g. Church [2]).

Generally, parsing was rather slow, considering the shortness of the sentences. Disambiguation certainly had a positive impact on parsing time, e.g. the ratio of sentences parsed in less than ten seconds rose from 6% to about 40%.

4.3 Analysis of data 2

Nodis and *Small* were in trouble due to excessively slow parsing. The first 9 sentences were parsed by all three setups. Here are the relevant statistics.

	Nodis		Small		Eng	
S	parses	time(sec)	parses	time	parses	time
1	92	1092	15	153	3	17
2	14	335	8	97	4	44
3	133	426	32	655	10	107
4	6	291	3	34	3	34
5	2	418	1	171	1	77
6	14	741	2	92	2	92
7	12	1878	4	428	4	429
8	13	3061	2	792	2	479
9	81	2021	34	1605	2	428

The general trend seems to agree with experiences from Data 1: the number of parses as well as parsing time generally decreases when more morphological disambiguation is carried out (however, note the curious exception in the case of sentence 3: parsing was faster with no disambiguation than with small disambiguation). Because of the scarcity of the data, more specific comparisons can not be made.

The setup with EngCG-2 disambiguation parsed all 200 sentences of Data 2. Because the other setups did not do this in the time allowed, no comparisons could be made. It may however be interesting to make two observations about the number of parses received. Consider the following table.

parses	sentences
0	3.5% (7)
1	18% (36)
1-5	28.5% (117)
1-20	86% (172)

Of all sentences, 96.5% got at least one parse, i.e. the slightly greater length of the input sentences does not seem to considerably affect the parser's coverage (the recognition rate was the same in Data 1).

The ambiguity rate increases considerably. For instance, only 28.5% of all sentences in Data 2 (compared to the 80% of Data 1) received 1-5 parses.

4.4 Analysis of data 3

In the analysis of the 20-word sentences, even the setup using the EngCG-2 disambiguator was in trouble: within the time allowed, the system analysed only 25 sentences. All of them received at least one parse.

5 Discussion and conclusion

We have investigated the use of two rule-based morphological disambiguators – the grammar of one developed over several years, the grammar of the other written in one day – for facilitating syntactic parsing in a nontrivial finite-state parser, paying special attention to the following issues:

- the possible negative effects of morphological disambiguation to the parser's recognition rate,
- whether morphological disambiguation can contribute to resolution of syntactic ambiguity, and
- the effect of morphological disambiguation on parsing time

On the basis of empirical tests, two encouraging observations can be made. Firstly, the parser's recognition rate was not considerably impaired by either disambiguator. The argued strength of rule-based disambiguators is their high recall (partly at the expense of precision); the present observations seem to support the argument. Secondly, especially the EngCG-2 disambiguator contributed to syntactic disambiguation (i.e. the number of parses decreased considerably). This is probably due to the lexicalised nature of the disambiguation grammar.

Our observations about the effect of morphological disambiguation on parsing time are somewhat mixed. Obviously, both disambiguators made parsing faster; for example, the EngCG-2 disambiguator made the parsing of the 200 10-word sentences about six times faster than with no morphological disambiguation. However, these experiments do not encourage use of morphological disambiguators if they are expected to make parsing possible; a parser troubled by long sentences should be improved with other techniques as well. One possible extension of the present study is to apply computationally cheap CG-style disambiguation techniques also to syntactic ambiguities before using the finite-state parser.

Acknowledgements

The parsing software used in these experiments was written by Pasi Tapanainen ([9, 10]).

References

1. E. Charniak, G. Carroll, J. Adcock, A. Cassandra, Y. Gotoh, J. Katz, M. Littman and J. McCann. 1996. Taggers for parsers. *Artificial Intelligence*, Vol. 85, No. 1-2.
2. K. W. Church. 1992. Current Practice in Part of Speech Tagging and Suggestions for the Future. In Simmons (ed.), *Sbornik praci: In Honor of Henry Kučera, Michigan Slavic Studies.* Michigan. 13-48.
3. F. Karlsson. 1990. Constraint Grammar as a Framework for Parsing Running Text. In H. Karlgren (ed.), *Proc. Coling'90.* Helsinki.
4. F. Karlsson, A. Voutilainen, J. Heikkilä and A. Anttila (eds.). 1995. *Constraint Grammar. A Language-Independent System for Parsing Unrestricted Text.* Berlin and New York: Mouton de Gruyter.
5. K. Koskenniemi. 1983. *Two-level Morphology. A General Computational Model for Word-form Production and Generation.* Publications 11, Department of General Linguistics, University of Helsinki.
6. K. Koskenniemi. 1990. Finite-state parsing and disambiguation. *Proc. Coling'90.* Helsinki, Finland.
7. Oflazer, K. and I. Kuruöz. 1994. Tagging and morphological disambiguation of Turkish text. *Procs. ANLP-94.*
8. C. Samuelsson and A. Voutilainen. 1997. Comparing a linguistic and a stochastic tagger. *Proc. EACL-ACL97.*. ACL, Madrid.
9. P. Tapanainen. 1992. "Äärellisiin automaatteihin perustuva luonnollisen kielen jäsennin" (A finite state parser of natural language). Licentiate thesis. Dept. Computer Science, University of Helsinki.
10. P. Tapanainen. 1996. *The Constraint Grammar Parser CG-2.* Dept. General Linguistics, University of Helsinki.

11. P. Tapanainen. 1997. Applying a finite-state intersection grammar. in Emmanuel Roche and Yves Schabes, editors, *Finite-state language processing.* A Bradford Book, MIT Press, Cambridge, Massachusetts. 311-327.
12. A. Voutilainen. 1994. *Three studies of grammar-based surface parsing of unrestricted English text.* (Doctoral dissertation.). Publications 24, Dept. General Linguistics, University of Helsinki.
13. A. Voutilainen. 1995. A syntax-based part of speech analyser. *Proc. EACL'95.* Pages 157-164.
14. A. Voutilainen. 1997. The design of a (finite-state) parsing grammar. in Emmanuel Roche and Yves Schabes, editors, *Finite-state language processing.* A Bradford Book, MIT Press, Cambridge, Massachusetts.
15. A. Voutilainen and P. Tapanainen. 1993. Ambiguity Resolution in a Reductionistic Parser. *Proc. EACL'93.* ACL, Utrecht. Pages 394-403.
16. A. Voutilainen and T. Järvinen. 1995. Specifying a shallow grammatical representation for parsing purposes. *Proc. EACL'95.* ACL, Dublin.
17. O. Wauschkuhn 1995. The influence of tagging on the results of partial parsing in German corpora. *Proc. Fourth International Workshop on Parsing technologies.* Prague/Karlovy Vary, Czech Republic, September 1995. Pages 260-270.