Grammar-based language technology for the Sámi languages

Trond Trosterud

This talk presents the Sámi language technology project in some detail. Often, language technology projects are either commercial (and hence closed for inspection), or they are small and run with no explicit infrastructure. This is our contribution to a concrete discussion on how to run medium-scale, decentralised, open-source language technology projects for minority languages.

Introduction

In this talk I present a practical framework for grammar-based language technology for minority languages in some detail. These matters are seldom the topic of discussion; we tend to go directly to the scientific results. But in order to obtain these results, one needs a good project infrastructure. Moreover, for minority languages the bottleneck is often the lack of human expertise, persons with knowledge both of the language, of linguistics, and of language technology. In such situations we need to organise the work in order to facilitate cooperation and to avoid duplication of the work. Although our model hardly is the ultimate one, it is the result of an accumulation of experience from different types of projects; commercial, academic and grass root open-source, and we hereby present it as a possible source of inspiration.

The Sámi languages make up one of the seven subbranches of the Uralic language family, Finnish and Hungarian being the most well known members of two of the other subbranches. Typologically, the Sámi languages have many properties in common with the other Uralic languages, but several non-segmental morphological processes have entered the languages as well. There are 6 Sámi literary languages, North, Lule, South, Kildin, Skolt and Inari Sámi. All of them are written with the Latin alphabet (including several additional letters) except Kildin Sámi, which is written with the Cyrillic alphabet.

Prior to our project, the main focus within Sámi computing had been the localisation issue. 4 of the 6 Sámi languages have letters not found in the Latin 1 (or Latin 2) code table. At present, this issue is more or less solved, and North Sámi is the language with fewest speakers that at the same time is localised, out of the box, on all 3 major operating systems. No other language technology applications existed prior to our work.

Project status quo, goals and resources

The work is organised in two projects, with slightly different goals. It started out as a university-based project, with a goal of building a morphological parser and disambiguator for North, Lule and South Sámi, in order to use them for scientific purposes: Making a tagged corpus with a web-based graphical interface, and using it for syntactic, morphological and lexical research, publishing reverse dictionaries, etc. In 2003 the Norwegian Sámi parliament wanted advice on how to build a Sámi spellchecker. They saw the construction of this tool as vital for the use of North Sámi as an administrative language. As a

result of this, we are now 3 persons working on the University project and 4 ½ people working on the Sámi parliament project. The projects will run with the present financing for another 2 years.

Status quo is that we have a parser with a recall of 80 - 93 % on grammatical analysis of words in running text (modulo genre), and we disambiguate the morphological output with a recall of 99 % and a precision of 93 %, and slightly worse outcome for syntactic analysis. The parsers behind these results contain 99 morphophonological rules, 573 continuation lexica and 1855 disambiguation rules.

The figures below shows output for the morphological parser of the sentence *Mii háliidit muitalit dan birra* "We would like to tell about it".

```
0%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>100%
"<Mun>"
        "mun" Pron Pers Sg1 Nom
"<háliidan>"
        "háliidit" V TV PrfPrc
        "háliidit" V TV Actio Acc
        "háliidit" V TV Actio Gen
        "háliidit" V TV Actio Nom
        "háliidit" V TV Ind Prs Sg1
"<muitalit>"
        "muitalit" V TV Inf
        "muitalit" V TV Ind Prt Sg2
        "muitalit" V TV Ind Prs Pl1
        "muitalit" V TV Ind Prs Pl3
"<dan>"
        "dat" Pron Dem Sg Acc
        "dat" Pron Dem Sg Gen
        "dat" Pron Pers Sg3 Acc
        "dat" Pron Pers Sg3 Gen
"<birra>"
        "birra" Pr
        "birra" Po
"<.>"
        "." CLB
```

Figure 1 Morphological analysis of a Sámi sentence

Figure 2 shows the same sentence in disambiguated mode. Here, all irrelevant morphological readings are removed, and in addition, syntactic information has been added, on the basis of the information given by the morphological disambiguator.

```
0%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>100%

252 sets (using 962 different tags)
239 mappings
27 constraint sections
1624 constraint rules
"<Mun>"
        "mun" Pron Pers Sg1 Nom @SUBJ
"<háliidan>"
        "háliidit" V TV Ind Prs Sg1 @+FAUXV
"<muitalit>"
        "muitalit" V TV Inf @-FMAINV
"<dan>"
        "dat" Pron Pers Sg3 Gen @GP>
"<birra>"
        "birra" Po @ADVL
"<.>"
        "." CLB <<<
```

Figure 2 Disambiguated version of the same sentence

As for the speller project, we have an alpha version, made with the *aspell* utility. The parser has been put to use in interactive pedagogical programs, and there are concrete plans for making a Sámi text-to-speech application.

Choice of approach

Grammatical vs. statistical approach

We use a grammar-based, rather than a statistical approach (proponents of the statistical approach often refer to this dichotomy as a choice between a "symbolic" and a "stochastic" approach), which means that our parsers rely on a set of grammar-based, manually written rules, that can be inspected and edited by the user. There are several reasons for our choice:

1. We think some of the prerequisites for good results with the statistical approach are not present in the Sámi case.
2. We want our work to produce grammatical insight, not only functioning programs.
3. Overall, we think the grammatical approach is better.

Ad 1: Good achievements with a statistical approach require both large corpora, and a relative simple morphological structure (low wordform / lemma ratio), as is the situation for English. Sámi and many other languages have a rich morphological structure and a paucity of corpus resources, whereas the basic grammatical structure of the languages is reasonably well understood.

Ad 2: Our work is a joint academic and practical project. Work on minority languages will typically be carried out as cooperation projects between research institutions and in-group individuals or organisations devoted to the strengthening of the languages in question. Whereas private companies will look at the ratio of income to development cost, and care less about the

developmental philosophy, it is important for research institutions to work with systems that are not ``black boxes", but that are able to give insight into the language beyond merely producing a tagger or a synthetic voice.

Ad 3: We are convinced that grammar-based approaches to both parsing and machine translation are superior to the statistical ones. Studies comparing the two approaches, such as Chanod and Tapanainen 1994, support this conclusion.

This does not mean that we rule out statistical approaches. In many cases, the best results will be achieved by combining grammatical and statistical approaches. A particularly promising approach is the use of weighted automata, where frequency considerations are incorporated into the arcs of the transducers. Use of standalone statistical methods we would like to apply after the grammatical analysis must give in. In other words, the cooperation should be ruled by the motto *don't guess if you know*.

Choosing between a top-down and a bottom-up approach

Within grammatical approaches to parsing, there are two main approaches, which we may brand top-down and bottom-up. The top-down approaches try to map a possible sentence structure upon the sentence, as a possible outcome of applying generative rules on an initial S node. If successful, the result is a syntactic tree displaying the hierarchical structure of the sentence in question.

The bottom-up approach, on the contrary, takes the incoming wordforms and the set of their possible readings as input. Then they disambiguate multiple readings based upon context, and build structure.

We chose a bottom-up approach because it was robust, it was able to analyse any input, and it gave good results.

Linguistic tools

The tools behind our morphological analyser

For our morphological analyser, we build finite-state transducers, and we use the finite-state tools provided by Xerox, and documented in Beesley and Karttunen 2003. For morphophonological analysis, there is a choice of using the parallel, two-level morphology model, dating back to Koskenniemi 1983, with *twolc*, or the sequential model, presented in Karttunen et al.1992, with *xfst*. Xerox' advice is to use the latter, we use the former, but we see this mainly as a matter of taste. The morphophonological and lexical tools are composed into a single transducer during compilation, as described in the literature. Cf. the figure below.

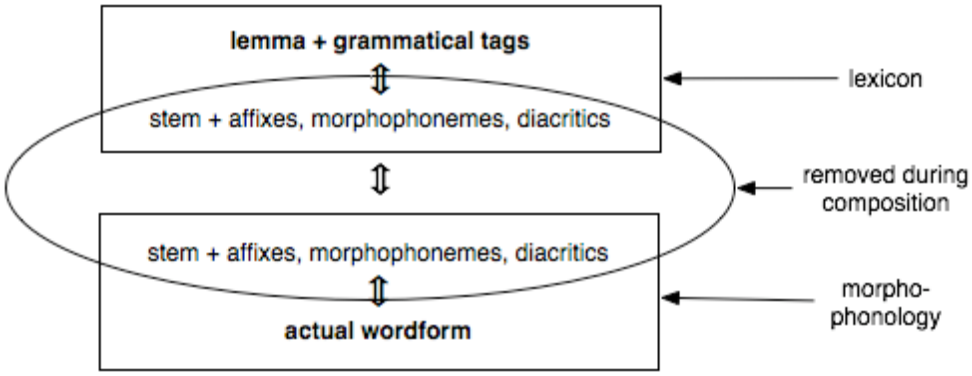

Figure 3 A schematic overview of the lexicon and morphophonology of the parser.

A more serious question is the choice of Xerox tools vs. open source tools. In our project, we have no wish to modify the source code of the rule compilers themselves, but we notice that all binary files compiled by the *xfst*, *lexc* and *twolc* compilers are copyrighted property of the Xerox Corporation. Is it as if you have written your own C program, but the compiled version of your program is copyright-owned by Kernighan and Ritchie, the authors of the C compiler. This much being said, it has been a pleasure working with Xerox, they have been very helpful, and as they see no commercial potential in Sámi, we notice no practical consequences of the fact that all our parsers are marked "Copyright Xerox corporation".

The tools behind our disambiguator

For disambiguating the output of the morphological transducer, we use constraint grammar. This is a framework dating back to Karlsson 1990, and the leading idea is that for each wordform of the output, the disambiguator looks at the context, and removes any reading that does not fit that context. The last reading can never be removed, and in the successful case, only the appropriate reading is left. The Brill tagger can be seen as a machine-learning variety of the constraint grammar parser.

There are several versions of the constraint grammar compilers. The original one was written in Lisp by Fred Karlsson. Later, Pasi Tapanainen wrote a compiler in C, called CG-2, this version may be licensed from www.connexor.com. We use an open source version of this compiler, made by Eckhard Bick. It must be stressed that the debugging facility of the Connexor compiler is superior to its competitors.

The optimal implementation would probably be to write the constraint grammar as a finite state transducer, as suggested in the Finite State Intersection Grammar framework. So far, nothing has come out of this work.

One-base, multi-purpose parsers

Working with minority languages, the lack of human resources is often as hard a problem as the lack of financial ones. With this in mind, avoiding duplicate work becomes crucial. The most time-consuming task in any linguistic project is to build and maintain a lexicon, be it in the form of a paper dictionary, a

semantic wordnet, or the lexicon for a parser. The optimal solution is to keep only one version of the lexicon, and to extract relevant information from it, in order to automatically build paper and electronic dictionaries, orthographical wordlists, or parsers. In our project, this has not yet been implemented, but for new languages we try out prototype models to make this work for new languages. Our plan is to use xml as text storage, and various scripts to extract the relevant lexicon versions.

It goes without saying that we use one and the same source for morphological transducer for linguistic analysis, pedagogical programs, spellers, etc. These applications often need slightly different transducers, in which case we mark the source code so that it is possible to compile different transducers from the same source code. For the academic project we make a tolerant parser, that analyses as much of the attested variation as possible. The spellchecker has a totally different goal, here we build a stricter version, that only accepts the forms codified in the accepted standard. This approach is even more appropriate as we are the only language technology project working on Sámi. Any further application will build upon our work, and our goal is to make it flexible enough to make that possible.

Infrastructure

Computer platform

Our project is run on Linux and Mac OS X (Unix). The Xerox tools come in a Windows version also, but the lack of a descent command-line environment and automatic routines for compiling makes it unpractical to use Windows. The cvs base is set up on a central Linux machine; otherwise we use portable Macintoshes, both because they have a nice interface, and because they offer programs that make it easier to work from different locations, such as the SubEthaEdit program mentioned below.

Character set and encoding

Most commercially interesting languages are covered by one of the 8-bit ISO standards. Very many minority languages fall outside of this domain. It is our experience that it is both possible and desirable to use UTF-8 (multi-byte Unicode) in our source code, i.e. to build the parser around the actual orthography of the language in question, rather than to construct some auxiliary ASCII representation. With the latest versions of the Linux and Unix operative systems and shells, we have access to tools that are UTF-8 aware, and although it takes some extra effort to tune the development tools to multi-byte input, the advantage is a more readable source code (with correct letters instead of digraphs) and an easier input/output interface, as UTF-8 now is the de facto standard for digital publishing.

There is one setting where one could consider using a transliteration, and that is for languages using syllabic scripts, such as Inuktitut and Cherokee. If you have a rule saying that a final vowel is changed in a certain environment, a syllabic script will not give you any single vowel symbol to change, rather that changing, say *a* to *e* in a certain context, your rule must change syllabic symbol BA to BE, DA to DE, FA to FE, GA to GE, etc. It still may be better to use the original orthography, though; each case requires its own evaluation process.

Directory structure

We have put some effort in finding a good directory structure for our files. The philosophy as as follows: Different types of files are kept separate. The source files have their own directory, binary and developer files are kept separate.

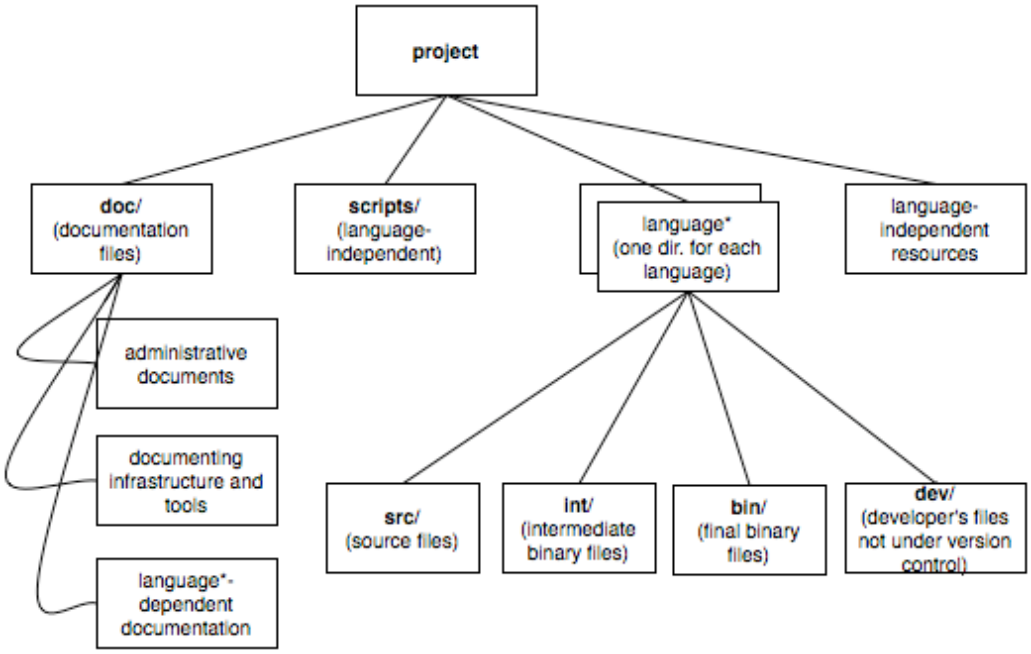

Figure 4 Directory structure

Version control

All our source and documentation files are under version control using cvs. This means that the original files are stored on our central computer (with backup routines), and that each co-worker *checks out* a local copy that becomes his or her version to work on. After editing, the changed files are then copied back, or *checked in* to the central repository. For each check-in, we write a short note telling what we have done. We also have set up a forwarding routine, so that all co-workers get a copy of all cvs log messages via mail.

```
----------------------------
revision 1.63
date: 2005/05/24 14:23:14;  author: maren;  state: Exp;  lines: +2 -1
Extended the rule "Optional Vowel Shortening after Short 1st Syllable" to work a
lso for sequences where j:i is the first consonant in the consonant cluster.
----------------------------
revision 1.62
date: 2005/04/13 06:43:21;  author: sjur;  state: Exp;  lines: +2 -2
Minor editorial change.
----------------------------
revision 1.61
date: 2005/04/01 13:40:01;  author: trond;  state: Exp;  lines: +2 -2
Added some instructive Y1 symbols to some of the examples following the rules.
----------------------------
revision 1.60
date: 2005/03/07 14:20:11;  author: trond;  state: Exp;  lines: +5 -0
Changed rule "Stem vowel in Past Tense...", by adding reference to j:, and a ref
erence to Y6 as well as to Y7.
----------------------------
revision 1.59
date: 2005/03/07 09:10:21;  author: thomas;  state: Exp;  lines: +1 -1
Have done nothing.
----------------------------
```

Figure 5 Quote from cvs log

Using cvs (or some other version control system) is self-evident to any programmer, and it may be seen as quite embarrassing that such a trivial fact is even mentioned here. It is our experience that the use of version control systems is by no means standard within academic projects, and we strongly urge anyone not using such tools to consider doing so. Backup routines become easier, and when growing from one-person projects to large projects, it is a prerequisite for being able to have several co-workers collaborating on the same source files. We will even recommend cvs for one-person projects. Using cvs, it is easier to document what has been done earlier, and to go back in previous versions to find out when a particular error may have crept in.

Building with make

Another self-evident programmer's tool is the use of makefiles, via the program *make*. In its basic form, *make* functions like a script, and saves the work of typing the same long set of compilation commands again and again. With several source files, it becomes important to know whether one should compile or not. *make* keeps track of the age of the different files, and compiles a new set of binary files only when the source files are newer then the target binary files. The picture shows the dependencies between the different source and binary files.

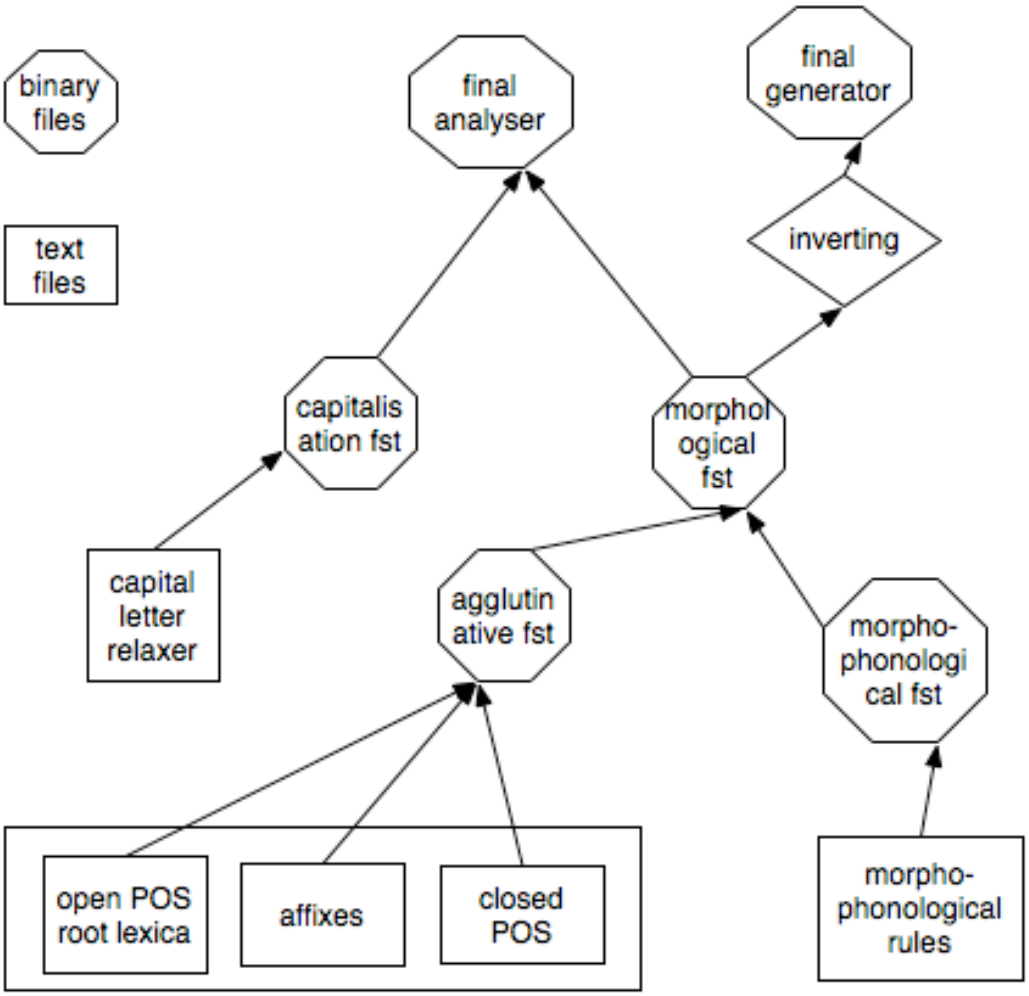

Figure 6 Dependencies in the project's Makefile

Tracing bugs

As the project grows, so does the number of people debugging it, and thereby the number of bugs and errors. We have designed an error database, in our case *Bugzilla*, which keeps track of the errors. The database can be found at (the temporal) address *http://129.242.176.176/giellatekno/bugzilla/*, the stable url will be *http://giellatekno.uit.no/bugzilla/*. People interested may visit the url. There is a requirement that you log in with an e-mail account and (preferably) a name, but otherwise the bug database is open for inspection.

Internal communication in a decentralised project

We have co-workers in Tromsø, Kautokeino and Helsinki. Crucial for the project's progress is the possibility of coordinating our work. For that, we have the following means:

- We have made a project-internal newsgroup. Discussion is carried out there rather in personal emails, since more than one person may have

something to say on the issue, and since it is easier to go back to earlier discussions using the newsgroup format

- For simultaneous editing of the same document, be it source code or a meeting memo, we use a program called *SubEthaEdit* (*http://www.codingmonkeys.de/subethaedit/* - only for Mac OS X). This program makes it possible for several users to edit the same file at the same time. Combined with telephone (or voice chat!), we may discuss complicated matters on a common rule set while editing together, even though we sit in different countries.
- For informal discussions, we use chat programs. The built-in Mac OS X chat application iChat also facilitates audio and video chats with decent to high quality of the video and sound (mainly restricted by the available bandwidth)
- We have meetings over the phone, although we planned to conduct them using iChat (up to ten participants in the same audio chat); technical problems with a firewall has stopped us from this, though
- The cvs version control and Bugzilla error database also facilitate working in several locations

Documentation

In our experience, a systematic approach to documentation is required when the project engages only one worker, and it is indispensable when the number of workers grows beyond two. Working on the only Sámi language technology project in the world, we acknowledge that all future work will take our work as a starting point. We thus work in a hundred-years perspective, and write documentation so that people following us will be able to read what we have done.

We document:

- The external tools we use (with links to the documentation provided by the manufacturer)
- The infrastructure of our project
- Our source files: the linguistic decisions we have made,

In an initial phase, we wrote the documentation in html, and it was available only internally on the project machines. Now, we write the documentation in xml, and convert it to html via the xml publishing framework Forrest, cf *http://forrest.apache.org/*. Documentation can be published in many ways, but it is our experience that it is convenient to read the documentation in a hypertext format such as html. As the documentation has grown we also use a search engine to find what we have written on a given topic, which Forrest provides.

The internal documentation of our project is open for inspection, at the web site http://divvun.no/ (the proofing tools project) as well as http://giellatekno.uit.no (the academic disambiguator project). The technical documentation is written in English, and it can be found under the tab *Teknihkalaš dok*. By publishing the documentation we make it easy to explain to others what we do, and we hope that it will inspire others, and perhaps give us some constructive feedback as well. The only possible drawback by this openness is that it exposes our weaknesses to the whole world. So far, we have not noticed any negative effects in this regard.

Costs

Except for the computers themselves and the operating system and applications that come with them, we have mostly used free or open-source software for all our tasks. In the few cases where we have paid for software, there are free or open-source alternatives. The notable exception is the set of compilers for morphophonological automata. For analysing running text and generating stray forms, the Xerox tools can be used in the versions found in Beesley and Karttunen 2003. For our academic project, these tools have proven good enough, but in order to generate larger paradigms, the commercial version of the tools is needed.

As for the computers, the only really demanding task is compiling the transducers. If one is willing to wait a few moments more, any recent computer can do fine. Macs turned out to be a good choice for our projects, and the cheapest Mac can be bought for roughly 500 USD/EUR. One good investment, though, is more RAM, preferably nothing less than 1 GB.

Summary

When doing language technology for minority languages, we are constantly faced with the fact that there are few people working with each language, and that different language projects set off in different directions, often due to coincidences. Our answer to this challenge is to share both our experiences and our infrastructure with others. By doing this, we hope that people will borrow from us and comment upon what we do and how we do it. We also look forward have a look at other solutions, and to borrowing improvements back.

References

Beesley, Kenneth R. and Lauri Karttunen 2003: *Finite State Morphology*. Stanford: CSLI Publications. (cf. http://www.fsmbook.com/)

Bick, Eckhard 2000: *The Parsing System "Palavras". Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework*. Dr.Phil. Thesis, University of Århus

Brill, Eric, 1992: A Simple Rule-based Part of Speech Tagger. *Proceedings of the Third Conference on Applied Natural Language Processing*, ACL, Trento, Italy, 1992

Chanod and Tapanainen, 1994: Tagging French - comparing a statistical and a constraint-based method. *Seventh Conference of the European Chapter of the Association for Computational Linguistics* 149-156.

Jelinek, Frederick 2004. *Some of my best friends are linguists*. LREC 2004. http://www.lrec-conf.org/lrec2004/doc/jelinek.pdf

Karlsson, Fred 1990: Constraint Grammar as a Framework For Parsing Running Text. Hans Karlgren, editor, *Papers presented to the 13th International Conference on Computational Linguistics*, volume 3, pages 168–173, Helsinki, Finland, August. ICCL, Yliopistopaino, Helsinki.

Lauri Karttunen, Ronald M. Kaplan, and Annie Zaenen. 1992. Two-level morphology with composition. In *COLING'92*, pages 141-148, Nantes, France, August 23-28.

Koskenniemi, Kimmo 1983: *Two-level Morphology: A General Computational Model for Word-form Production and Generation*. Publications of the Department of General Linguistics, University of Helsinki.

Samuelsson, Christer and Atro Voutilainen. 1997: Comparing a linguistic and a stochastic tagger. 35th Annual Meeting of the Association for Computational Linguistics, 1997.

Voutilainen, Atro, Juha Heikkilä and Arto Anttila 1992: *Constraint Grammar of English, A performance-Oriented Introduction,* Publication No. 21, Department of General Linguistics, University of Helsinki,