A Robust Shallow Parser for Swedish

Ola Knutsson, Johnny Bigert, Viggo Kann
Numerical Analysis and Computer Science
Royal Institute of Technology, Sweden
{knutsson, johnny, viggo}@nada.kth.se

Abstract

In this paper, a robust parser for Swedish is presented. The parser identifies the internal structure of phrases, but does not build full trees. In addition to phrase identification, clause boundaries are detected. The parser is designed for robustness against noisy and ill-formed data. An evaluation on 15 000 words shows that the parser's accuracy on phrase bracketing is 88.7 per cent and the F-score for clause boundary identification is 88.3 per cent.

1 Introduction

In many NLP-applications, the robustness of the internal modules of an application is a prerequisite for the success and usability of the system. The term robustness is a bit unclear and vague, but in NLP, it is often used in the sense robust against noisy, ill-formed, and partial natural language data. The full spectrum of robustness is defined by Menzel (1995), and further explored according to parsing in (Basili and Zanzotto, 2002). In the following, we will focus on a parser developed for robustness against ill-formed and partial data, called Granska Text Analyzer (GTA).

2 Shallow Parsing

Shallow parsing is becoming a strong alternative to full parsing, see e.g. (Li and Roth, 2001) due to its robustness and quality. Shallow parsing can be seen as a parsing approach in general, but also as pre-processing for full parsing. It is not one technique, rather a collection of techniques including hand-crafted rule based methods and systems based on machine learning. The main idea is to parse only parts of the sentence and not build a connected tree structure and thus limiting the complexity of the analysis. The partial analysis is well suitable for modular processing which is important in a system that should be robust (Basili and Zanzotto, 2002). A major initiative in shallow parsing came from Abney (1991), arguing both for psycholinguistic evidence for shallow parsing and also its usability in applications for real world text or speech. Abney used hand-crafted cascaded rules implemented with finite state transducers. Current research in shallow parsing is mainly focusing on machine learning techniques (Hammerton et al., 2002).

An initial step in shallow parsing is often called text chunking, i.e. dividing the sentence into base level phrases. The Swedish sentence *Den mycket gamla mannen gillade mat* (*The very old man liked food*) would be chunked as:
(NP Den mycket gamla mannen)(VC gillade)(NP mat)
The next step after chunking is often called phrase bracketing. Phrase bracketing means analyzing the internal structure of the base level phrases (chunks). Many researchers have focused on NP bracketing e.g. (Tjong Kim Sang, 2000). The same sentence as above will be bracketed with internal structure of the phrases:
(NP Den (AP mycket gamla) mannen)(VC gillade)(NP mat)

What internal phrases that should be assimilated with other more high-level phrases is a question for debate, and also how complex a phrase could be, for instance is *Gamla stans bokhandel* one phrase or should it by bracketed like [NP Gamla stans] [NP bokhandel]? These questions and others make it hard to compare different parsers with one another. The only way to compare is to use the same annotated test data, a tree bank. The chosen bracketing depends on the relation to a specific syntactic theory or the needs in real world applications. Some shallow parsers do also include some analysis of grammatical functions (subject, main verb, object etc.).

3 Parsers for Swedish

Most parsers for Swedish are surface oriented, and designed for unrestricted text. Early initiatives on parsing Swedish focused on the usage of heuristics (Brodda, 1983) and surface information as in the Morp Parser (Källgren, 1991). The Morp was also designed for parsing using very limited lexical knowledge.

A more full syntactic analysis is accomplished by the Uppsala Chart Parser (UCP) (Sågvall Hein, 1982). UCP has been used in several applications, for instance in machine translation (Sågvall Hein et al., 2002).

Two other parsers, have been developed recently. One uses machine learning (Megyesi, 2002) while the other is based on finite-state cascades, called Cass-Swe (Kokkinakis and Johansson-Kokkinakis, 1999). Notable is that Cass-Swe also assigns functional information to constituents.

There is also a deep parser developed in the Core Language Engine (CLE) framework (Gambäck, 1997). The deep nature of this parser limits its coverage.

Furthermore, two other parsers identify dependency structure using Constraint Grammar (Birn, 1998) and Functional Dependency Grammar (Voutilainen, 2001). These two parsers are also commercialized. The Functional Dependency parser actually builds a connected tree structure, where every word points at a dominating word.

4 A Robust Shallow Parser for Swedish

The Granska Text Analyzer is rule-based and relies on hand-crafted rules written in a formalism with a context-free backbone. The rules are augmented with features. It is quite often claimed that the grammars of shallow parsers are quite large, containing thousands of rules (Hammerton et al., 2002). This is not the case with GTA. In total GTA contains 260 rules. 200 of these rules identify different kinds of phrases, 40 rules are disambiguation rules that select heuristically between ambiguous phrase identifications. Clause boundaries are identified with 20 rules. However, the number of rules is not the only aspect of grammar complexity. Interaction between rules and recursion are also important aspects of grammar complexity.

In a first phase, the parser selects grammar rules top-down and uses a passive chart. The rules in the grammar are applied on part-of-speech tagged text, either from an integrated tagger or from an external source. GTA identifies constituents and assigns phrase labels. However, no full trees with a top node are built.

The disambiguation of phrase boundaries is in a first phase done within the rules, and secondly using heuristic selection. In a third phase, a disambiguation and selection algorithm called the Tetris algorithm is applied to the remaining ambiguities.

The analysis is surface-oriented and identifies many types of phrases in Swedish. The basic phrase types are adverb phrases (ADVP), adjective phrases (AP), infinitive verb phrases (INFP), noun phrases (NP), prepositional phrases (PP) and limited verb phrases and verb chains (VC). The internal structure of the phrases is parsed when appropriate and the heads of the phrases are identified. PP-attachment is left out of the analysis since the parser does not include a mechanism for resolving PP-attachments.

4.1 Basic Phrase Categories in GTA

The selection of phrase categories is based on the needs in rule based and statistical grammar checking (Bigert and Knutsson, 2002). When a Swedish standard for phrase bracketing is present (i.e. a treebank), GTA will be converted to it. Some important changes in the phrase bracketing will also be done based on the evaluation below. Most work in the

development of GTA focused on the noun phrases. Noun phrases are often difficult to identify correctly, but also very important in many applications.

- Noun Phrases (NP)

  The identification of noun phrases includes minimal noun phrases e.g. *en liten bil* (*a little car*), proper names like *Peter Forsberg*, and pronouns e.g. *jag* (*I*). Complex noun phrases with apposition (e.g. *min vän generalen* (*my friend the general*) and coordinated NPs like *långa spelare och tuffa backar* (*tall players and tough backs*) are also identified. Complex noun phrases are bracketed as one noun phrase including two noun phrases. Relative clauses are attached to the NP, e.g. *mannen som står därborta* (*the man that stands over there*) is identified as one NP, but prepositional phrases are not included in the noun phrase. In the next version of GTA, no post-modifying phrases will be included in the noun phrases to make the phrase bracketing more consistent and transparent.

- Verb Chains and limited Verb Phrases (VC)

  Simple verb chains like *har spelat* (*has played*) and more complex verb phrases like *har mannen inte spelat* (*has the man not played*) are identified by GTA.

- Prepositional Phrases (PP)

  Only non-recursive prepositional phrases are identified, which means that *mannen på bänken i parken* (*the man on the bench in the park*) is identified as two prepositional phrases. The general prepositional phrase includes a preposition followed by a noun phrase, e.g. *i det gamla huset* (*in the old house*).

- Adverb Phrases (ADVP)

  Adverb phrases are singleton adverbs e.g. *snart* (*soon*) or a group of adverbs *så långt norrut* (*that far north*).

- Adjective Phrases (AP)

  Adjective phrases are simple groups of adjectives e.g. *lilla röda* (*little red*) or coordinated adjectives *liten och röd* (*small and red*).

- Infinitive Verb Phrases (INFP)

  All infinitive verb phrases that are identified begin with the infinitive marker and are followed by the infinitive verb and an optional NP. Examples of infinitive verb phrases that are identified by GTA are *att sjunga* (*to sing*) and *att spela fotboll* (*to play soccer*).

4.2 Clause Boundary Detection

The detection of clause boundaries is an important step in sentence processing. Dividing the sentence into clauses limits the complexity of the sentence. In addition to the parsing of phrase structure, clause boundaries (CLB) are detected in GTA, resembling Ejerhed's algorithm for clause boundary detection (Ejerhed, 1999). Ejerhed's rules for clause boundary detection are implemented in a straightforward manner following the patterns pointed out in Ejerhed's paper. A few new rules have been developed. Totally, 20 rules for clause boundary detection are used in the parser.

The output from the parser is given in the so-called `IOB` format (Ramshaw and Marcus, 1995). See Figure 1 for a sentence with phrase labels and clause boundaries in the `IOB` format.

As an example, the word *kraftfulla* (*powerful*) in the sentence in figure 1 was tagged with the `IOB` tags APB, NPB and PPI which means that the word *kraftfulla* begins (B) an adjective phrase (AP) and noun phrase (NP) and is inside (I) a prepositional phrase (PP). Some words/tokens in the sentence are outside the phrases and are therefore assigned the tag O (outside).

4.3 Robustness against ill-formed and Fragmentary Natural Language Data

The parser was designed for robustness against ill-formed and fragmentary sentences. One task for the parser is to analyze text from second language learners and other text types which include different kinds of errors.

The parser is not facilitated with relaxation techniques, which is convenient in many systems (see e.g. (Jensen, 1993)). Instead the design of the parser follows the lines in the design of Constraint Grammar parsing (Karlsson et al., 1995) and also Functional Dependency parsing (Järvinen and

```
Vi (we)                NPB           CLB
har (have)             VCB           CLI
inga (no)              NPB           CLI
pengar (money)         NPI           CLI
och (and)              O             CLB
vi (we)                NPB           CLI
kan (can)              VCB           CLI
inte (not)             ADVPB|VCI     CLI
finansiera (finance)   VCI           CLI
vår (our)              NPB           CLI
verksamhet (business)  NPI           CLI
utan (without)         PPB           CLI
kraftfulla (powerful)  APB|NPB|PPI   CLI
besparingar (savings)  NPI|PPI       CLI
,                      O             CLB
hävdar (claims)        VCB           CLI
han (he)               NPB           CLI
.                      0             CLI
```

Figure 1: *Example sentence showing the* `IOB` *format.*

Tapanainen, 1997) – the question of grammaticality is not dealt with within the parser. Grammaticality is more used as a reason for the selection of one interpretation prior to another. In addition to the noise in textual data, there is also a rich source for errors from the internal modules of the parsing system, e.g. tokenization and tagging errors. Robust parsers must handle these internal errors, or at least degrade gracefully.

As an example, agreement is not considered in noun phrases and predicative constructions (Swedish has a constraint on agreement in these constructions). By avoiding the constraint for agreement, the parser will not fail due to textual errors or tagging errors. In other words, the parser does not decide about the grammaticality in such constructions. Tagging errors that do not concern agreement are to some extent handled using a set of tag correction rules based on heuristics on common tagging errors.

Another important design feature of the parser is that no top node is built. Only local trees are built, and there is no interaction between the rules for different phrase types, e.g. the rules for NP recognition are not interacting with the rules that identify verb chains. The final selection of the internal structure of the local trees is not done within the grammar; instead, a special module takes care of this work, thereby limiting the complexity of the grammar and keeping the parser efficient.

4.4 Modularization: to Disambiguate or not to Disambiguate?

One interesting question in parsing is at what stage the program should disambiguate. Should a module disambiguate with the information at hand or should it leave some ambiguity to the next modules? Voutilainen (1994) argues for the value of dealing with both morphological, clause boundary, and syntactical ambiguities in the same rule. This requires a lexical approach with information actually including the wanted parse.

We have chosen to disambiguate as completely as possible. The input to the parser is part-of-speech tagged text, with only one tag assigned to each word. But at the same time it is still possible in the rules to use textual data and also alternatives rejected by the tagger. To conclude, the basic case in GTA is fully disambiguated data, but text matchings and alternative morphosyntactic tagging can be used in the grammar rules when appropriate, for instance to handle systematic tagging errors. The output from the parser is fully disambiguated, but internally alternative parses are always available. Modularization is thus the choice of GTA, but the modules can interact with each other partly bi-directionally, which means that low level rules (e.g. tagging correction) can interact with the ambiguous syntactic level, but not with the disambiguated surface syntactic level.

4.5 Different Kinds of Rules

The rules in GTA are written in a partly object-oriented notation resembling Java or C++. An example rule, NPmin below, has two parts separated with an arrow. The first part contains a matching condition. The second part specifies the action that is triggered when the matching condition is fulfilled.

Each line in the first part of the rule contains an expression that must evaluate to true in the matching rule. This expression may be a general Java expression, another rule or a feature value (matching text, lemma, word class, or grammatical feature).

The action part of the rule states that the rule is a so called help rule (possibly recursive function), which may be used by other rules. In addition, the feature values of the whole phrase or pattern are assigned.

In the example, the action is triggered when a determiner (determiners, not including “denna”, “dessa” and “denne” (this/these)) is followed by an optional adverb or a cardinal number, followed by another token with the word class adjective, ordinal number or participle (optional), followed by a noun. The reason for excluding “denna”, “dessa” and “denne” is that these determiners set the feature value for species of the NP to definite.

The noun is identified by the rule NN, which matches nouns that are fully recognized by the tagger, the rule also identifies and more important assigns feature values to nouns that are only partly recognized by the tagger. It is important to notice that NPmin contains several rules separated by the operator **;** which means logical **or** between rules. In the example of NPmin below, two rules are presented. The first rule matches constructions like *den lilla bilen* but also the errorneous NP *den liten bil*. There is no constraint for agreement between for instance the adjective and noun in this rule. The second rule in NPmin detects only NPs without initial determiners. Thus, the first disambiguation of phrase boundaries is done in this first basic rule. The rule uses the limited context-sensitive abilities of the rule language in GTA. Without the power of context sensitive rules the parser will end up with several analyses even on simple NPs.

If there are no feature values in the part-of-speech tagged data, the rule NN_NO_TAGS looks at the left context of the noun, and assigns the values from preceding token if the preceding word seems to belong to the same NP. In rule NPmin the feature values are taken from the noun, but as seen in rule NN_NO_TAGS the feature values are taken from the context.

4.6 Selecting the Constituent Structure

Heidorn and Jensen (Jensen et al., 1983) developed an algorithm for dealing with ill-formed and fragmentary sentences, called parse fitting. Parse fitting is used when the parser has failed to analyze a sentence using a conventional grammar. The fitting algorithm is implemented as a set of rules, that chooses a head constituent, and then the remaining constituents are fitted in. The selection is based on linguistic preference, i.e. first is a VP with tense and subject chosen. If such a VP is not found a VP with tense but no subject is selected. After that, phrases

```
NPmin@
{
X((wordcl=dt & text!="denna" &
   text!="dessa" & text!="denne"
   & text!="detta")
   | wordcl=hd | wordcl=rg),
X2(wordcl=ab | wordcl=rg)?,
Y(wordcl=jj | wordcl=ro | wordcl=pc)*,
(NN/Z)()
-->
action(help, wordcl:=Z.wordcl, pnf:= undef,
       gender:=Z.gender, num:=Z.num,
       spec:=Z.spec, case:=Z.case)
;

X(wordcl!=dt & wordcl!=hd),
---endleftcontext---,
X2(wordcl=ab | wordcl=rg),
Y(wordcl=jj | wordcl=ro | wordcl=pc)+,
(NN/Z)()
-->
action(help, wordcl:=Z.wordcl, pnf:= undef,
       gender:=Z.gender, num:=Z.num,
       spec:=Z.spec, case:=Z.case)
;

...
}
```

```
NN@
{
X(wordcl=nn & gender!=undef &
  num!=undef & spec!=undef & case!=undef)
-->
action(help, wordcl:=nn, gender:=X.gender,
        num:=X.num, spec:=X.spec, case:=X.case)
;
(NN_NO_TAGS/X)()
-->
action(help, wordcl:=nn, gender:=X.gender,
        num:=X.num, spec:=X.spec, case:=X.case)
}
```

```
NN_NO_TAGS@
{
X(wordcl=dt | wordcl=hd | wordcl=ps |
        wordcl=jj | wordcl=ro),
endleftcontext,
Z(wordcl=nn & gender=undef &
        num=undef & spec=undef & case=undef)
-->
action(help, wordcl:=nn, gender:=X.gender,
        num:=X.num, spec:=X.spec, case:=nom)
;
...
}
```

without verbs (NPs, PPs) are chosen and so forth. If this head constituent does not cover the entire sentence, remaining constituents are added on either side of the head constituent based on another preference. The fitting procedure works outward from the head constituent.

The Tetris Algorithm

One main difference between GTA and Heidorn and Jensens approach is that GTA never tries to build full tree from a core grammar. GTA always make a parse fitting procedure, by doing so many ambiguity and efficiency problems are avoided. GTA's approach to parse fitting is not linguistically motivated, instead it relies on longest matching. The constituents are sorted according to length. Then the longest constituent is selected from the right to the left. The fitting procedure then tries to fit in the second longest constituent to the left, to the right and inside the selected constituent and so forth. Overlapping constituents cannot be selected. Thus, the whole sentence will be assigned a constituent structure, and in addition, the internal strucuture of the constituents is filled in when a shorter constituent can be fitted in a longer constituent.

5 Evaluation

The parser has been evaluated on 15 000 words from the SUC corpus. Five text genres were used. In the absence of a Swedish treebank annotated with constituency trees, the texts were manually annotated with constituency structure, without top-nodes, based on the output from the parser. However, the manual annotation is more homogenous across the phrase types than the output of GTA. This means that there are systematic errors in the output from the parser. The evaluation results are therefore calculated on the untuned output from the parser. The accuracy on the phrase structure task is 88.7 per cent (see table 1) and the F-score for the clause boundary detection is 88.2 per cent (see table 2). In the evaluation we used part-of-speech tagged data from four different sources/taggers: a baseline tagger called Unigram, which chooses the most frequent tag for a given word and the most frequent tag (for open word classes) for unknown words, the original corpus tags from SUC (Ejerhed et al., 1992), a faster version of the Brill tagger, called fnTBL (Ngai and Florian, 2001) and the hidden Markov model (HMM) tagger TnT (Brants, 2000) were used in the evaluation.

The parser seems to work best on PPs, APs, VCs and NPs (see table 3). Adverb phrases and infinitive verb phrases are identified with a lower accuracy. It is often hard for the rules to determine the end of these constructions. Some noun phrases are identified with post attributes as relative clauses, the results are not fully satisfying, and therefore one refinement of GTA should be to exclude all post-modifying phrases from the analysis. For a more detailed description of the evaluation see (Bigert et al., 2003).

In addition to the standard evaluation described above, a glass-box evaluation of GTA's robustness was made (Bigert et al., 2003). In this evaluation spelling errors were automatically introduced in the texts, and fed to the parsing system. The evaluation showed that GTA is robust, and degrades gracefully, i.e. GTA degrades linearly with the part-of-speech taggers' degradation. In other words, if the tagger is robust (i.e. predictable), GTA will also be robust.

6 Concluding Remarks and Future Work

Without a Swedish tree bank the results of the evaluation are preliminary, they can only serve as an indicator of the parser's performance. The choices made when annotating the test corpus are important when evaluating a parser. When there is a Swedish treebank available, more reliable and easy comparable evaluations of GTA[1] can be made.

The next step in the development of GTA is to extend the analysis to clause types and syntactic functions. With syntactic functions included in the analysis, GTA can be compared not only with parsers assigning constituency structure, but partly with dependency parsers as well.

[1] GTA can be tested here:http://skrutten.nada.kth.se/grim/form.html

Tagger	$Accuracy$
UNIGRAM	81.0
BRILL	86.2
TNT	88.7

Table 1: *Accuracy in per cent from the parsing task. Parsing based on the on the manual tagging in SUC had 88.4% accuracy. A baseline parser using the original SUC tagging had 59.0% accuracy. For a given part-of-speech tag the baseline parser assigns the most frequent parse for that tag.*

Tagger	$F-score$
UNIGRAM	84.2
BRILL	87.3
TNT	88.3

Table 2: *F-score from the clause boundary identification task. Identification based on the original SUC tagging had an F-score of 88.2%. A baseline identifier had an F-score of 69.0%. The baseline identifier assigns CLB to the first word of each sentence and CLI to the other words.*

Type	$Accuracy$	Count
ADVP	81.9	1008
AP	91.3	1332
INFP	81.9	512
NP	91.4	6895
O	94.4	2449
PP	95.3	3886
VC	92.9	2562
Total	88.7	

Table 3: *F-scores for the individual phrase categories from the parse task.* TNT *was used to tag the text.*

References

S. Abney. 1991. Parsing by chunks. In R. C. Berwick, S. P. Abney, and C. Tenny, editors, *Principle-Based Parsing: Computation and Psycholinguistics*, pages 257–278. Kluwer Academic Publishers, Boston.

R. Basili and F. M. Zanzotto. 2002. Parsing engineering and empirical robustness. *Natural Language Engineering*, 8(2–3):97–120.

J. Bigert and O. Knutsson. 2002. Robust error detection: A hybrid approach combining unsupervised error detection and linguistic knowledge. In *Proc. 2nd Workshop Robust Methods in Analysis of Natural language Data (ROMAND'02), Frascati, Italy*, pages 10–19.

J. Bigert, O. Knutsson, and J. Sjöbergh. 2003. Automatic evaluation and robustness and degradation in tagging and parsing. In *Proc. RANLP 2003*, pages 51–57, Borovets, Bulgaria.

J. Birn. 1998. Swedish constraint grammar. Technical report, Lingsoft Inc, Helsinki, Finland.

T. Brants. 2000. Tnt – a statistical part-of-speech tagger. In *Proc. 6th Applied NLP Conference, ANLP-2000*, Seattle, USA.

B. Brodda. 1983. An experiment with heuristic parsing of Swedish. In *Proc. of First Conference of the European Chapter of the Association for Computatlona Linguistics*, pages 66–73, Pisa, Italy.

E. Ejerhed, G. Källgren, O. Wennstedt, and M. Åström. 1992. *The Linguistic Annotation System of the Stockholm-Umeå Project*. Department of Linguistics, University of Umeå, Sweden.

E. Ejerhed. 1999. Finite state segmentation of discourse into clauses. In A. Kornai, editor, *Extended Finite State Models of Language*, chapter 13. Cambridge University Press.

B. Gambäck. 1997. *Processing Swedish Sentences: A Unification-Based Grammar and some Applications*. Ph.D. thesis, The Royal Institute of Technology and Stockholm University.

J. Hammerton, M. Osborne, S. Armstrong, and W. Daelemans. 2002. Introduction to special issue on machine learning approaches to shallow parsing. *J. Machine Learning Research*, Special Issue on Shallow Parsing(2):551–558.

T. Järvinen and P. Tapanainen. 1997. A dependency parser for English. Technical report, Department of Linguistics, University of Helsinki.

K. Jensen, G. Heidorn, L. Miller, and L. Ravin. 1983. Parse fitting and prose fixing: getting a hold on ill-formedness. *American Journal of Computational Linguistics*, 9(3–4):147–160.

K. Jensen. 1993. PEG: The PLNLP English grammar. In K. Jensen, G. E. Heidorn, and S. D. Richardson, editors, *Natural Language Processing: The PLNLP Approach*, pages 29–43. Kluwer, Boston, USA.

G. Källgren. 1991. Parsing without lexicon: the morp system. In *Proc. Fifth Conference of the European Chapter of the Association for Computational Linguistics*, pages 143–148, Berlin, Germany.

F. Karlsson, A. Voutilainen, J. Heikkilä, and A. Anttila. 1995. *Constraint Grammar. A Language Independent System for Parsing Unrestricted text*. Mouton de Gruyter, Berlin, Germany.

D. Kokkinakis and S. Johansson-Kokkinakis. 1999. A cascaded finite-state parser for syntactic analysis of Swedish. In *Proc. 9th European Chapter of the Association of Computational Linguistics (EACL)*, pages 245–248, Bergen, Norway. Association for Computational Linguistics.

X. Li and D. Roth. 2001. Exploring evidence for shallow parsing. In Walter Daelemans and Rémi Zajac, editors, *Proc. of CoNLL-2001*, pages 38–44, Toulouse, France.

B. Megyesi. 2002. Shallow parsing with PoS taggers and linguistic features. *J. Machine Learning Research*, Special Issue on Shallow Parsing(2):639–668.

W. Menzel. 1995. Robust processing of natural language. In *Proc. 19th Annual German Conference on Artificial Intelligence*, pages 19–34, Berlin. Springer.

G. Ngai and R. Florian. 2001. Transformation-based learning in the fast lane. In *Proceedings of NAACL-2001*, pages 40–47, Carnegie Mellon University, Pittsburgh, USA.

L. Ramshaw and M. Marcus. 1995. Text chunking using transformation-based learning. In David Yarovsky and Kenneth Church, editors, *Proc. Third Workshop on Very Large Corpora*, pages 82–94, Somerset, New Jersey. Association for Computational Linguistics.

A. Sågvall Hein, A. Almqvist, E. Forsbom, J. Tiedemann, P. Weijnitz, L. Olsson, and S. Thaning. 2002. Scaling up an mt prototype for industrial use. Databases and data flow. In *Proc. Third International Conference on Language Resources and Evaluation (LREC 2002)*, pages 1759–1766, Las Palmas, Spain.

A. Sågvall Hein. 1982. An experimental parser. In *Proc. of the Ninth International Conference on Computational Linguistics (Coling 82)*, pages 121–126, Prague.

E. F. Tjong Kim Sang. 2000. Noun phrase representation by system combination. In *Proc. ANLP-NAACL 2000*, Seattle, Washington, USA.

A. Voutilainen. 1994. Designing a parsing grammar. Technical report, Department of Linguistics, University of Helsinki, Finland.

A. Voutilainen. 2001. Parsing Swedish. In *Proc. 13th Nordic Conference on Computational Linguistics (Nodalida-01)*, Uppsala, Sweden.