Compiling and Using Finite-State Syntactic Rules

Kimmo Koskenniemi, Pasi Tapanainen and Atro Voutilainen

University of Helsinki
Research Unit for Computational Linguistics
Hallituskatu 11
SF-00100 Helsinki
Finland

Abstract

A language-independent framework for syntactic finite-state parsing is discussed. The article presents a framework, a formalism, a compiler and a parser for grammars written in this formalism. As a substantial example, fragments from a nontrivial finite-state grammar of English are discussed.

The linguistic framework of the present approach is based on a surface syntactic tagging scheme by F. Karlsson. This representation is slightly less powerful than phrase structure tree notation, letting some ambiguous constructions be described more concisely.

The finite-state rule compiler implements what was briefly sketched by Koskenniemi (1990). It is based on the calculus of finite-state machines. The compiler transforms rules into rule-automata. The run-time parser exploits one of certain alternative strategies in performing the effective intersection of the rule automata and the sentence automaton.

Fragments of a fairly comprehensive finite-state grammar of English are presented here, including samples from non-finite constructions as a demonstration of the capacity of the present formalism, which goes far beyond plain disambiguation or part of speech tagging. The grammar itself is directly related to a parser and tagging system for English created as a part of project SIMPR[1] using Karlsson's CG (Constraint Grammar) formalism.

1. Introduction

The present finite-state approach to syntax should not be confused with eg. attempts to characterize syntactic structures with regular phrase structure grammars. Instead of using trees as a means of representing structures, we use syntactic tags associated with words, and the finite-state rules constrain the choice of tags. This style of representation was adopted from Karlsson's CG approach and an earlier Finnish parser called FPARSE (Karlsson 1985, 1990).

The current approach employs a shallow surface oriented syntax. We expect it to be useful in syntactic tagging of large text corpora, information retrieval, and as a starting point for more elaborate syntactic or semantic analysis.

1.1 Representation of sentences

We represent the sentences as *regular expressions*, or equivalently, as *finite-state networks*, which list all combinatory possibilities to interpret them. Consider the sentence:

the program runs.

A (simplified) representation for the morphologically processed but syntactically unanalyzed sentence as a regular expression could be roughly as follows:

```
@@
 the  DEF ART
[@ | @/ | @< | @>]
[[program N NOM SG [@SUBJ | @OBJ |
                    @PREDC]] |
 [program V PRES NON-SG3 @FINV @MAINV] |
 [program V INF]]
[@ | @/ | @< | @>]
[[run V PRES SG3 @FINV @MAINV] |
 [run N NOM PL [@SUBJ | @OBJ | @PREDC]]]
@@
```

Here `@@` represents a sentence boundary, `@` a word boundary, `@/` an ordinary clause boundary, `@<` a beginning of a center embedded clause, and `@>` the end of such an embedding.

Square brackets '[...]' are used for grouping, and vertical bars '|' separate alternatives. Each word has been assigned all possible syntactic roles it could assume in sentences (eg. `@SUBJ`

1. Esprit II project No. 2083, Structured Information Management: Processing and Retrieval.

or @OBJ or @PREDC). Note that between each two words there might be a clause boundary or a plain word boundary. The regular expression represents a number of strings (some 320) which we call the *readings* of the (unanalyzed) sentence. The following is one of them:

```
@@ the DEF ART @/
   program V PRES NON-SG3 @FINV @MAINV @
   run N NOM PL @PREDC @@
```

This one is very ungrammatical, though. It will be the task of the rule component to exclude such, and leave only the grammatical one(s) intact:

```
@@ the DEF ART @
   program N NOM SG @SUBJ @
   run V PRES SG3 @FINV @MAINV @@
```

Note that in this framework, the parsing does not build any new structures. The grammatical reading is already present in the input representation.

1.2 The role of rules

The task for the rules here is (as is the case with the CG approach by Karlsson) to:

- exclude those interpretations of ambiguous words which are not possible in the current sentence,
- choose the correct type of boundaries between each two words, and
- determine which syntactic tags are the appropriate ones.

Rules should preferably express meaningful constraints which result in the exclusion of all ungrammatical alternatives. Each rule should thus be a grammatical statement which effectively forbids certain tag combinations.

Rules in the CG formalism are typically dedicated for one of the above tasks, and they are executed as successive groups.

In finite-state syntax, rules are logically unordered. Furthermore, in order to achieve word level disambiguation, one typically uses rules which describe the occurrences of boundaries and syntactic tags in *grammatically correct* structures rather than indicating how the incorrect interpretations can be identified. Thus, the three *effects are achieved*, even if individual finite-state rules cannot be classified into corresponding three groups.

1.3 Rule automata

Finite-state rules are represented using regular expressions and they are transformed into finite-state automata by a rule compiler.

The whole finite-state grammar consists of a set of rules which constrain the possible choices of word interpretations, tags and boundaries to only those which are considered grammatical. The entire grammar is effectively equivalent to the (theoretical) intersection of all individual rule automata. However, such an intersection would be impractical to compute due to its huge size.

The logical task for any finite-state parser in the current approach is to compute the intersection of the unanalyzed sentence automaton and each rule automaton. Actual parsing can be done in several alternative ways which are guaranteed to yield the same result, but which vary in terms of efficiency.

2. The finite-state rule formalism

Tapanainen (1991) has implemented a compiler and a parser for finite-state grammars. The compilation and the parsing is based on a Common Lisp finite-state program package written by him. Tapanainen also reports in his Master's thesis (1991) new methods for optimizing the result of the compilation and improving the speed of parsing.

The current rule compiler has only few built-in rules or definitions. Instead, it has a formalism for defining relevant expressions and new rule types. There are two types of definitions for this purpose. The first one defines a constant regular expression which can later on be referred to by its name:

name = *expression*;

Some basic notations are defined in this way such as the *dot* which stands for a sequence of tokens within a single word:

```
. = \[@@ | @ | @/ | @< | @>];
```

The *backslash* '\' denotes any sequence of tokens not containing occurrences of its argument (which here lists all types of word and clause boundaries). A variation of the dot is a *dot-dot* '..' which represents a sequence of tokens within the same clause:

```
@<> = @< [. | @ | @/]* @>;
 .. = [. | @ | @<>]*;
```

The second type of definitions has parameters, and it can be used for expressions which vary according to their values:

name(*param*$_1$, .., *param*$_n$) = *expression*;

The *expression* is a regular expression formulated using constant terms and the parameter symbols *param*$_i$. An example of this type of definitions is the following which requires every clause to be of a given form X:

```
clause(X) = \ [ [@@ | @/ | @<]
                -[X | -..]
                [@> | @/ | @@]];
```

The formula forbids subsequences which are clauses but not of form X (the middle term is easier to understand as [-X & ..]).

Experience with writing actual large scale grammars within the finite-state framework has indicated that we need more flexibility in defining rules than what was first expected. This flexibility is achieved by having one very general rule format:

```
expression;
```

The expression simply defines a constraint for all sentences, ie. it is already as such equivalent to a rule automaton. Forbidding unwanted combinations or sequences, such as two finite verbs within the same clause, can be excluded eg. by a rule:

```
UNIQUE(FINV);
```

Here, UNIQUE is a definition which has been made using the formalisms above, and is available for the grammar writer. Using the UNIQUE definition, one can express general principles, such as that there is at most one main verb, at most one subject etc. in each clause.

Most of the actual rules still use the right arrow format:

```
expression =>
     left-context _ right-context;
```

All three parts of the rules are regular expressions. The rule requires that any occurrence of *expression* must be surrounded by the given context.

3. English finite-state grammar

The English finite-state grammar discussed here was written by Voutilainen. The grammar itself is much more comprehensive than what can be described in this paper. Although the grammar already covers most of the areas of English grammar that it is intended to cover, it is still far from complete in details. The grammar, when complete, will be part of Voutilainen's PhD dissertation (forthcoming). This section presents some general principles from that grammar, and a few examples from more complex phenomena.

3.1 Goals of the grammar

The present grammar has many goals and characteristics similar to those of the SIMPR Constraint Grammar:

- the ability to parse unrestricted running texts with a large dictionary,
- concrete, surface-oriented description in terms of dependency syntax.

The current finite-state syntax uses, indeed, the same ENGTWOL lexicon as the SIMPR CG syntax (Karlsson et al. 1991). The set of syntactic features are adopted from the CG description almost as such with a few additions.

In the present finite-state approach, however, we aim at:

- more general and linguistically motivated rules (fewer, more powerful and general rules in the grammar),
- more accurate treatment of intrasentential structure (three types of clause boundaries instead of one), and
- a satisfactory description of certain complex constructions and sentence structures.

The present formalism can achieve somewhat more general and powerful rules than the current CG formalism through the use of full regular expression notation.

3.2 Clause boundaries

Some power and accuracy is gained through a commitment to use a notation for clause boundaries which is exact in defining when words belong to the same or a different clause. The two formalisms are equivalent in many cases:

```
@@ The dog chased a cat
@/ which ate the mouse @@
```

The more elaborate clause boundary marking makes a difference in case of center-embedding:

```
@@ The man @< who came first @> got the job @@
```

This convention indicates that there are two clauses:

```
The man .. got the job
.. who came first ..
```

3.3 Constituent structure

Head-modifier relations are expressed (here and in the CG) with tags, eg.:

```
a DET @DN>
big A @AN>
cat N @SUBJ
```

The head of a NP is tagged as a major constituent, here as a subject. In case the constituent is a coordinated one, each of the coordinated head gets the same tag:

```
John's N GEN @GN>
brother N NOM SG @SUBJ
and COORD @CC
aunts N NOM PL @SUBJ
```

The genitival attribute @>GN modifies *at least* the next noun (brother) but possibly also some further ones at the same level of coordination (aunts).

3.4 An example

Let us consider the following (classical) sentence

Time flies like an arrow.

The input to the finite-state syntax comes from the ENGTWOL morphological analyzer with some modifications and extensions in the sets of features associated with words:

```
@@
[[time N NOM SG
     [@<P | @NN> | @APP |@PCOMPL-O/N/-F |
      @PCOMPL-O/N | @PCOMPL-S/N/-F |
      @PCOMPL-S/N | @I-OBJ/-F | @I-OBJ |
      @OBJ/-F | @OBJ | @SUBJ/-F | @SUBJ ]] |
 [time <SVO>
     [[V IMP VFIN @+FMAINV] |
      [V INF [@<NOM-FMAINV |
              @-FMAINV/-F | @-FMAINV]]]]]
[@ | @/ | @< | @>]
[[fly <SVO> <SV> V PRES SG3 VFIN @+FMAINV]
 [fly N NOM PL
     [@<P | @APP | @PCOMPL-O/N/-F |
      @PCOMPL-O/N | @PCOMPL-S/N/-F |
      @PCOMPL-S/N | @I-OBJ/-F | @I-OBJ |
      @OBJ/-F | @OBJ | @SUBJ/-F | @SUBJ]]]
[@ | @/ | @< | @>]
[[like PREP [@<NOM | @ADVL | @ADVL/INV]] |
 [like N NOM SG
     [@<P | @NN> | @APP | @PCOMPL-O/N/-F |
      @PCOMPL-O/N | @PCOMPL-S/N/-F |
      @PCOMPL-S/N | @I-OBJ/-F | @I-OBJ |
      @OBJ/-F | @OBJ | @SUBJ/-F | @SUBJ]]
 [like <SVOC/A> <SVO> <SV> V
     [[SUBJUNCTIVE VFIN @+FMAINV] |
      [IMP VFIN @+FMAINV] |
      [INF  [@<NOM-FMAINV | @-FMAINV/-F |
             @-FMAINV] |
      [PRES NON-SG3 VFIN @+FMAINV]]]]
[@ | @/ | @< | @>]
[an <Indef> DET CENTRAL ART SG @DN>]
[@ | @/ | @< | @>]
[[arrow V [[IMP VFIN @+FMAINV] |
           [INF [@<NOM-FMAINV |
                 @-FMAINV/-F | @-FMAINV]]]]
 [arrow N NOM SG
     [@<P | @NN> | @APP | @PCOMPL-O/N/-F |
      @PCOMPL-O/N | @PCOMPL-S/N/-F |
      @PCOMPL-S/N | @I-OBJ/-F | @I-OBJ |
      @OBJ/-F | @OBJ | @SUBJ/-F | @SUBJ]]]
@@
```

This small sample sentence representation contains some 21 million readings.

Each syntactic-function label starts with @. Many of the common labels like @SUBJ have been replaced by the combination of @SUBJ/-F and @SUBJ to reflect the distinction of subjects of non-finite constructions from those of the main verb. A similar distinction is made in the verbal entries.

The grammar is committed to exclude only those readings which are ungrammatical. Thus, several readings may pass the rules, in this case, the following six:

```
1. @@ time N NOM SG @NN> @
   fly N NOM PL @SUBJ @
   like <SVOC/A> <SVO> <SV>
        V PRES NON-SG3 VFIN @+FMAINV @
   an <Indef> DET CENTRAL ART SG @DN> @
   arrow N NOM SG @OBJ @@

2. @@ time N NOM SG @SUBJ @
   fly <SVO> <SV> V PRES SG3 VFIN @+FMAINV @
   like PREP @ADVL @
   an <Indef> DET CENTRAL ART SG @DN> @
   arrow N NOM SG @<P @@

3. @@ time N NOM SG @SUBJ @
   fly <SVO> <SV> V PRES SG3 VFIN @+FMAINV @
   like N NOM SG @OBJ @
   an <Indef> DET CENTRAL ART SG @DN> @
   arrow N NOM SG @APP @@

4. @@ time <SVO> V IMP VFIN @+FMAINV @
   fly N NOM PL @SUBJ @
   like <SVOC/A> <SVO> <SV>
        V PRES NON-SG3 VFIN @+FMAINV @
   an <Indef> DET CENTRAL ART SG @DN> @
   arrow N NOM SG @OBJ @@

5. @@ time <SVO> V IMP VFIN @+FMAINV @
   fly N NOM PL @OBJ @
   like PREP @<NOM @
   an <Indef> DET CENTRAL ART SG @DN> @
   arrow N NOM SG @<P @@

6. @@ time <SVO> V IMP VFIN @+FMAINV @
   fly N NOM PL @OBJ @
   like PREP @ADVL @
   an <Indef> DET CENTRAL ART SG @DN> @
   arrow N NOM SG @<P @@
```

3.5 Overview of rules

The finite-state grammar for English consists of some 200 rules dedicated for several areas of the grammar:

- Internal structure of nominal and non-finite verbal phrases. The structure is described as head-modifier relations, including determiners, premodifiers and postmodifiers.
- Coordination at various levels of the grammar.
- Surface-syntactic functions of nominal phrases.

The structure of noun phrases is described using two approaches together. A coarse structure is fixed with the mechanism of definitions. It would not be feasible to use that mechanism alone (because it would lead to a context-free description). The definitions are supplemented with ordinary finite-state rules which enforce further restrictions.

3.6 Non-finite Constructions

Between the level of the nominal phrase and the finite clause, there is an intermediary level, that of *non-finite constructions* (see Quirk & al. 1985). These constructions resemble noun phrases when seen as parts of the surrounding clause because they act eg. as subjects, objects, preposition complements, etc., postmodifiers, or adverbials, eg.:

{Walking home} was wearisome.
She wants {to come}[1].
She was fond of {singing in the dark}.
The dog {barking in the corridor} was irritable.
{Tired by her journey}, she fell asleep.

Internally, non-finite constructions are like finite clauses because the main verb of a non-finite construction can have subjects, objects, adverbials etc. of its own.

Both finite and non-finite constructions have a *verbal skeleton*, which in a finite construction starts with a *finite verb* and ends with the first *main verb*. The finite verbal skeletons in the following examples are underlined:

She sings.
Will she be singing?
She would not have been singing unless ..

A non-finite verbal skeleton starts with certain kinds of non-finite verb (to+infinitive, present participle, past participle, non-finite auxiliary) and ends with the first main verb to the right:

It is easy to do it.
Tired by her journey, she went into her room.
They knew it all, having been there before.

Non-finite verb chains do not contain center-embedded verbs, whereas a non-finite construction can be center-embedded within a finite verb chain only if it is (a part of) a nominal phrase:

Can {shooting hunters} be dangerous?
Can men {shooting hunters} be dangerous?

The use of syntactic tags instead of a hierarchical tree-structure forces us to a very flat description of sentences. This might result in problems when describing clauses with non-finite constructions with a small set of tags, eg.:

The boy [kicking @MAINV] the [ball @OBJ]
[saw @MAINV] the [cow @OBJ].

A useful concept in clause-level syntax is the *uniqueness principle*. We wish to say, for instance, that in a clause, there is *at most one* (possibly co-ordinated) subject, object, or predicate complement. Uniqueness holds for the finite clause, and each non-finite construction separately, and this will be very difficult to formulate, if we use same tags for both domains (as in the above example).

The syntactic tags as given in the finite-state version of ENGTWOL capitalize heavily on non-finite constructions in order to overcome this problem:

The boy [kicking @MAINV/-F] the [ball @OBJ/-F]
[saw @MAINV] the [cow @OBJ].

Here, the object in the non-finite construction is furnished with a label different from the corresponding label used in the finite construction, so there is no risk of confusion between the two levels.

The duplication of certain labels for certain categories increases the amount of ambiguity, but, on the other hand, the new ambiguity seems to be of a fairly controllable type. The description of non-finite constructions boils down to two subtasks. One is to express constraints on the internal structure of non-finite constructions; the other, the control on their distribution.

In terms of verb chain and constituent structure, non-finite constructions resemble finite constructions. Their main difference is that word order in non-finite constructions is much more rigid.

We proceed with some examples of rules describing non-finite constructions. An infinitive acting as main verb in a non-finite construction is preceded by `to` acting as an infinitive marker or by a subject of a non-finite phrase or by a co-ordinated infinitive.

So we wish, for instance, the following utterances to be accepted:

He wants [to @INFMARK>] [go INF @-FMAINV/-F].
She saw [her @SUBJ/-F] [go INF @-FMAINV/-F].
She saw [her @SUBJ/-F]
[come INF @-FMAINV/-F] and [go INF @-FMAINV/-F].

The constraint is expressed as a rule:

```
!inf-main/-f =>
     [[@INFMARK> [@ !advl]*] |
      [!subj/-f !<*] |
      [!inf-main/-f !/-f* !phr-cc]] @ _ ;
```

Items preceded by an exclamation mark are constant definitions. `!/-f` signals any constituent that can occur in a postverbal position in a non-finite construction.

A past participle as a main verb in a non-finite construction must always be preceded by an appropriate kind of auxiliary or clause boundary.

1. There is another way to interpret this sentence without any non-finite constructions by including 'to come' in the finite verb chain. We have adopted the current interpretation in order to achieve certain linguistic generalizations.

For example:

[Having @-FAUXV/-F] [gone PCP2 @-FMAINV/-F] home, they rested.

This constraint corresponds to a rule:

```
!pcp2-main/-f =>
     [!prim-aux/-f | !clb] !advl* _ ;
```

There are further rules for the distribution of non-finite constructions with present participles, etc. Further rules have been written for the description of the internal structure of non-finite constructions which, in turn, is fairly straight-forward. The overall experience is that a fairly adequate description of these types of phenomena can be achieved by the set of syntactic tags proposed above accompanied by a manageable set of finite-state rules.

4. Implementation

We need a compiler for transforming the rules written in the finite-state formalism into finite-state automata, and a parser which first transforms sentences into finite-state networks, and then computes the logical intersection of the rule-automata and the sentence automaton.

4.1 Compilation of the rules

The grammar consisting of rules is first parsed and checked for formal errors using a GNU `flex` and `bison` parser generator programs. The rest of the compilation is done in Common Lisp by transforming the rules written in the regular expression formalism into finite-state automata.

Full-scale grammars tend to be large containing maybe a few hundred finite-state rules. In order to facilitate the parsing of sentences, the compiler tries to reduce the number of rule automata after each rule has been compiled. Methods were developed for determining which of the automata should be merged together by intersecting them (Tapanainen 1991). The key idea behind this is the concept of an *activation alphabet*. Some rule-automata turn out to be irrelevant for certain sentences, simply because the sentences do not contain any symbols (or combinations of symbols) necessary to cause the automaton to fail. Such rule-automata can be ignored when parsing those sentences. Furthermore, it turned out to be a good strategy to merge automata with similar activation alphabets (rather than arbitrary ones, or those resulting in smallest intersections).

4.2 Parsing sentences

The implementation of the parsing process is open to many choices which do not change the results of the parsing, but which may have a significant effect on the time and space requirements of the parsing. As a theoretical starting point one could take the following setup.

Parser A: Assume that we first enumerate all readings of a sentence-automaton. Each reading is, in turn, fed to each of the rule-automata. Those readings that are accepted by all rule-automata form the set of parses.

Parser A is clearly infeasible in practice because of the immense number of readings represented by the sentence-automaton (millions even in relatively simple sentences, and the number grows exponentially with sentence length).

A second elementary and theoretical approach:

Parser B: Take the sentence automaton and intersect with each rule-automaton in turn.

This is more feasible, but experiments have shown that the number of states in the intermediate results tends to grow prohibitively large when we work with full scale grammars and complex sentences (Tapanainen 1991). This is an important property of finite-state automata. All automata involved are reasonably small, and even the end result is very small, but the intermediate results can be extremely large (more than 100,000 states and beyond the capacity of the machines and algorithms we have).

A further refinement of the above strategy B would be to carefully choose the order in which the intersecting is done:

Parser C: Intersect the rule-automata with the sentence automaton in the order where you first evaluate each of the remaining automata according to how much they reduce the number of readings remaining. The one which makes the greatest reduction is chosen at each step.

This strategy seems to be feasible but much effort is spent on the repeated evaluation. It turns out that one may even use a one-time estimation for the order:

Parser D: Perform a tentative intersection of the sentence automaton and each of the rules first. Then intersect the rules with the sentence automaton one by one in the decreasing order of their capacity to reduce the number of readings from the *original* sentence automaton.

We may also choose to operate in parallel instead of rule by rule:

Parser E: Simulate the intersection of all rules and the sentence automaton by trying to enumerate readings in the sentence automaton but constraining the process by the rule-automata. Each time when a rule rejects the next token

proposed, the corresponding branch in the search process is abandoned.

This strategy seems to work fairly satisfactorily. It was used in the initial stages of the grammar development and testing together with two other principles:

- merging of automata into a smaller set of automata during the compilation phase using the activation alphabet of each automaton as a guideline
- excluding some automata before the parsing of each sentence according to the presence of tokens in the sentence and the activation alphabets of the merged automata.

Some further improvements were achieved by the following:

Parser F: Manually separate a set of rules defining the coarse clause structure into a phase to be first intersected with the sentence automaton. Then use the strategy E with the remaining rules. The initial step establishes a fairly good approximation of feasible clause boundaries. This helps the parsing of the rest of the rules by rejecting many incorrect readings earlier.

Parsing simple sentences like "time flies like an arrow" takes some 1.5 seconds, whereas the following fairly complex sentence takes some 10 seconds to parse on a SUN SPARCstation2:

> Nevertheless the number of cases in which assessment could not be related to factual rental evidence has so far not been so great as to render the whole system suspect.

The sentence automaton is small in terms of the number of states, but it represents some 10^{35} distinct readings.

5. Acknowledgments

The work of Tapanainen[1] is a part of the activity of the Research Unit for Computational Linguistics (RUCL) partly sponsored by the Academy of Finland. Voutilainen is a member of the SIMPR project at RUCL, sponsored by the Finnish Technology Development Center (TEKES). The SIMPR CG parser, grammars and dictionaries were designed and written by F. Karlsson, A. Voutilainen, J. Heikkilä, and A. Anttila. Many of these results and innovations are either directly used here, or have had a direct influence on the present results.

1. Electronic mail addresses of the authors are: Kimmo.Koskenniemi@Helsinki.FI, Pasi.Tapanainen@Helsinki.FI, avoutila@ling.helsinki.fi

6. References

E. Ejerhed, K. Church: Finite-State Parsing. F. Karlsson (ed.) *Papers from the Seventh Scandinavian Conference on Linguistics.* University of Helsinki, Department of General Linguistics, Publications, No. 10, pp. 410-432.

F. Karlsson 1985. Parsing Finnish in terms of Process Grammar. F. Karlsson (ed.) *Computational Morphosyntax: Report on Research 1981-84.* University of Helsinki, Department of General Linguistics, Publications, No. 13.

F. Karlsson 1990. Constraint Grammar as a Framework for Parsing Running Text. H. Karlgren (ed.) *COLING-90: Papers Presented to the 13th International Conference on Computational Linguistics.* Helsinki, Vol. 3, pp. 168-173.

F. Karlsson, A. Voutilainen, J. Heikkilä, A. Anttila, 7 January 1991. *Natural Language Processing for Information Retrieval Purposes.* SIMPR Document No. SIMPR-RUCL-1990-13.4e. Research Unit for Computational Linguistics, University of Helsinki, Finland. 220 pp.

F. Karlsson, A. Voutilainen, J. Heikkilä, A. Anttila (forthcoming), *Constraint Grammar: A Language-Independent System for Parsing Running Text.*

L. Karttunen, K.Koskenniemi, R. Kaplan 1987. A Compiler for Two-level Phonological Rules. *Tools for Morphological Analysis* (M. Dalrymple, R. Kaplan, L. Karttunen, K. Koskenniemi, S. Shaio, M. Wescoat). Center for the Study of Language and Information, Stanford. Report No. CSLI-87-108.

K. Koskenniemi 1983. *Two-level Morphology: A General Computational Model for Word-Form Recognition and Production.* University of Helsinki, Department of General Linguistics, Publications, No. 11. 160 pp.

K. Koskenniemi 1990. Finite-state Parsing and Disambiguation. H. Karlgren (ed.) *COLING-90: Papers Presented to the 13th International Conference on Computational Linguistics.* Helsinki, Vol. 2, pp. 229-232.

R. Quirk, S. Greenbaum, G. Leech, J. Svartvik 1985. *A Comprehensive Grammar of the English Language.* London, Longman.

P. Tapanainen 1991. *Äärellisinä automaatteina esitettyjen kielioppisääntöjen soveltaminen luonnollisen kielen jäsentäjässä.* ("Natural language parsing with finite-state syntactic rules.") Master's Thesis. Department of Computer Science, University of Helsinki.