CONTRIBUTIONS TO THE THEORY OF FINITE-STATE BASED GRAMMARS

Anssi YLI-JYRÄ

Academic dissertation to be publicly discussed, by due permission of the Faculty of Arts at the University of Helsinki, in auditorium XII, on the 20th of June, 2005, at 12 o'clock.

University of Helsinki
Department of General Linguistics
P.O. Box 9
FIN-00014 University of Helsinki
Finland

PUBLICATIONS
No. 38
2005

Author: Yli-Jyrä, Anssi M.
Title: Contributions to the Theory of Finite-State Based Linguistic Grammars
Type: Dissertation
Publisher: Department of General Linguistics, University of Helsinki
Address: P.O. Box 9, FIN-00014 University of Helsinki, Finland
Series: Publications of Department of General Linguistics, University of Helsinki
Number: 38

Supervisors:
– Professor Kimmo Koskenniemi
Department of General Linguistics, University of Helsinki, Finland
– Professor Lauri Carlson
Department of General Linguistics, University of Helsinki, Finland
Pre-Examiners:
– Professor Kemal Oflazer
Faculty of Engineering and Natural Sciences, Sabancı University, Turkey
– Adjunct Professor András Kornai
Budapest Institute of Technology, Hungary
Opponent: András Kornai
Custos: Kimmo Koskenniemi
Representative of the Faculty:
– Professor Arvi Hurskainen
Institute for Asian and African Studies, University of Helsinki, Finland

First printing, June 2005

Typeset by the author in LaTeX. The figures were drawn with *Xfig*, *xpic*, *graphviz* and *DGgraph*.

ISSN 0355-7170
ISBN 952-10-2510-7 (PDF) — http://ethesis.helsinki.fi
ISBN 952-10-2509-3 (paperback)
Helsinki University Printing House
Helsinki 2005

Abstract

This dissertation is a theoretical study of finite-state based grammars used in natural language processing. The study is concerned with certain varieties of *finite-state intersection grammars* (FSIGs) whose parsers define regular relations between surface strings and annotated surface strings. The study focuses on the following three aspects of FSIGs:

(i) Computational complexity of grammars under limiting parameters In the study, the *computational complexity* in practical natural language processing is approached through performance-motivated parameters on structural complexity. Each parameter splits some grammars in the *Chomsky hierarchy* into an infinite set of subset approximations. When the approximations are regular, they seem to fall into the *logarithmic-time hierarchy* and the *dot-depth hierarchy* of star-free regular languages. This theoretical result is important and possibly relevant to grammar induction.

(ii) Linguistically applicable structural representations Related to the *linguistically applicable representations* of syntactic entities, the study contains new bracketing schemes that cope with dependency links, left- and right branching, crossing dependencies and spurious ambiguity. New grammar representations that resemble the *Chomsky-Schützenberger* representation of context-free languages are presented in the study, and they include, in particular, representations for mildly context-sensitive non-projective dependency grammars whose performance motivated approximations are linear-time parseable.

(iii) Compilation and simplification of linguistic constraints *Efficient compilation methods* for certain regular operations such as the generalized restriction are presented. These include an elegant algorithm that has already been adopted as the approach in a proprietary finite-state tool. In addition to the compilation methods, an approach to on-the-fly *simplifications* of finite state representations for parse forests is sketched.

These findings are tightly coupled with each other under the theme of locality. I argue that the findings help us to develop better, linguistically oriented formalisms for finite-state parsing and to develop more efficient parsers for natural language processing.

Keywords: syntactic parsing, finite state automata, dependency grammar, first-order logic, linguistic performance, star-free regular approximations, mildly context-sensitive grammars

Preface

Efficient linguistic communication is our best interface to the rest of the world. It can give us a feeling of safety and identity. Meanwhile inefficiency in the communication tends to isolate people from each other, causing misunderstandings, insecurity and even wars. Efficient communication is, however, not free. For example in the European Union, where legislative documents are translated into some twenty official languages, a large number of professionals are needed in the translation business.

Computer-aided human translation and language learning could be used to reduce the economical costs of efficient cross-linguistic communication. In such systems, various computerized grammars play a crucial role, and especially computationally efficient grammars have received growing interest during the last few years. Many researchers hope to find ways to use various approximation methods together with their mathematical idealizations. This work is along this line of research.

Acknowledgements

I am especially thankful to Esa Nurro for introducing me to Apple II at the Seinäjoki Senior Secondary School when I was a 9-year old kid. I wish to thank Saara Järvinen (Kenya), Ritva Lehonkoski, Geoffrey Hunt, Johan Autio, and Harri Saarikoski for helping me to find a linguist within me. My idea of studying computational linguistics arose in 1980's when I had accidentally met a Finnish field linguist whose portable computer was equipped with a speech analyzer and a lexical database. Back then, I realized that computational linguistics is an important instrument for the tomorrow's globalized society where the living languages have to co-exists in digital form.

In 1990, I was a fresh student of computational linguistics at the University of Helsinki. Then, during the Pre-COLING tutorials that were given in Helsinki, Lauri Karttunen from Xerox gave a short, but very impressive introduction to Kimmo Koskenniemi's two-level model of finite-state morphology. When I later started my full-time studies after my military service, I studied under the guidance of professor Kimmo Koskenniemi. At his department, I witnessed the enthusiasm of the pioneers of the Constraint Grammar and Finite-State Intersection Grammar (FSIG).

In mid-1990's, I was responsible for language technology in a project that produced morpheme-level aligned bilingual texts for Ristin Voitto Publishing House. In the project, I made an attempt to use Tarja Heinonen's FSIG for Finnish, under the auspices of Krister Lindén at Lingsoft in 1994. Back then, FSIG was very slow in practice and I also encountered these problems with Finnish FSIG parser. As a reac-

tion, I wrote a heuristic, deterministic parser based on Tarja Heinonen's elegant FSIG rules in 1993-1994, but its grammar soon became too difficult to maintain because it was procedural.

Jussi Piitulainen, a co-student of mine, was courageous enough to try alternative approaches to FSIG parsing, and he told me about his locally tree-shaped automata in 1994. In the wake of his example, I also started to dream of new FSIG data structures and ended up with a largely different approach. My earliest article in this thesis is actually a short paper describing the essentials of the approach. When I had written it, Kimmo suggested that changing my master's thesis topic from bilingual alignment to finite-state parsing might be advisable, and I was willing to do so. Even now, ten year after the article, we feel that it still prompts further studies. During my early investigations for this thesis, Atro Voutilainen provided the source file for Eng-FSIG and Pasi Tapanainen made his parser available, and I am now deeply indebted to them.

The research behind this Ph.D. dissertation was made economically possible by the Ph.D. scholarship (ref.no. 010529) that I received from the language technology programme of the *Nordic Academy of Advanced Study* (NorFA) for the years 2002 – 2004. I am grateful to Henrik Holmboe in NorFA and Helge Dyvik for their personal concern and interest in my project. Besides NorFA, I am grateful for the considerable flexibility of my employer, the Finnish IT Centre for Science (CSC), where my bosses Leif Laaksonen and Manne Miettinen arranged my research leaves.

I am deeply indebted to my two supervisors whose instructions and interest in my research topic has been a tremendous encouragement and help: Professor Koskenniemi provided a sound, experienced view on finite-state algorithms and FSIG-related scenarios. Professor Lauri Carlson's vast knowledge on tense and logic was very useful when my adventurous ideas needed some pre-evaluation. I am also indebted to Magnus Steinby, Matti Nykänen, and Kerkko Luosto for instructing me in their expertise areas. I am thankful to Lauri Karttunen, André Kempe, and Ronald Kaplan for their help and reference concerning the restriction operation. I wish to thank G. Pullum, B. Watson, T. Lager, J. Nivre, A. Dikovsky, A. Joshi, B. Ravikumar, J. Brzozowski, P. Kolaitis, G. Morril, J. Kari, S. Yu, A. Maletti, U. Petersen, A. Okhotin, K. Lindén, K. Heljanko, and S. Turunen for helpful discussions. I am also thankful for my meetings with Maurice Gross when he still lived.

On a more personal level, I am grateful to Hanna W., Johanna R., and Kirsti K., Pirkko S., Orvokki H., Fred K., Martti V., Graham W., Matti M., Jan-Ola Ö., Martti N., and Arto A. for their support when I was a student, and to my co-students in KIT, ESSLLI, SIL and NorFA courses for the shared time.

Finally, my warmest thanks go to my parents and sisters for their belief in me. Through their loving example, I have been able to develop a trusting relationship to the Highest One, my Heavenly Father. I believe that He has given me the strength to complete this work.

Above all, I want to thank my dear wife, Heli, for her infinite patience to stand by me through my research process and my little sons, Risto and Juho, for bearing my long working days at home, and for their eagerness to play with me whenever possible.

A.Y.-J.

June 2005

Contents

List of Tables

List of Figures

List of Acronyms

AC . . . . . *arc consistency*

BDD . . . . . *binary decision diagram*

B-FSIG . . . . . *Bracketed finite-state intersection grammar*

BIT$(i : j)$ . . . . . *true if and only if bit j in the binary representation of i is 1* – a predicate for accessing extended variables in FO queries

CCG . . . . . *combinatory categorial grammar*

CFBG . . . . . *context-free bracketing grammar*

CFG . . . . . *context-free grammar*

CF-L-EDL-*S*-G . *context-free linear storage grammar with extended domain of locality*

CH . . . . . *Chomsky hierarchy* — a hierarchy of Types 0 - 3.

CL . . . . . *computational linguistics*

CLSG . . . . . *complicate locally, simplify globally*

CMLG . . . . . *colored multiplanar link grammar*

CNDG . . . . . *colored non-projective dependency grammar*

CP . . . . . *constraint propagation*

CRAM . . . . . *concurrent random access machine*

CSP . . . . . *constraint satisfaction problem*

DDH . . . . . *dot-depth hierarchy*

DFA . . . . . *deterministic finite automaton*

DFCA . . . . . *deterministic finite cover automaton*

DFS . . . . . *depth-first search*

DGP . . . . . *derivational generative power*

DL . . . . . *domain of locality*

ECFG . . . . . *extended context-free grammar*

EDL . . . . . *extended domain of locality*

Eng-FSIG . . . . . *English finite-state intersection grammar* – a specific FSIG constraint set designed by Atro Voutilainen in 1990's.

ESSLLI . . . . . *European Summer School in Logic, Language and Information*

FA . . . . . *finite automaton*

FCFBG . . . . . *flat context-free bracketing grammar*

FG . . . . . *formal grammar*

FL . . . . . *formal language*

FO . . . . . *first order boolean queries*

FO(DTC) . . . . . *first order queries with deterministic transitive closure*

FO(LFP) . . . . . *first order queries with least fixed point operator*

FO[$<$] . . . . . *first-order logic with linear order*

FO[$n^{O(1)}$] . . . . . *first order queries with BIT and polynomial number of extended variables*

FO[$(\log n)^{O(1)}$] . *first order queries with BIT and poly-logarithmic number of extended variables*

FO(TC) . . . . . *first order queries with transitive closure*

FS . . . . . *finite-state*

FSIG . . . . . *finite-state intersection grammar*

GES . . . . . *generative-enumerative syntax*

HGDG . . . . . *Hays-Gaifman dependency grammar*

IFSM . . . . . *incomplete finite-state machine*

LBB . . . . . *limited balanced bracketing*

LCFRS . . . . . *linear context-free rewriting system*

LF-AFA . . . . . *loop-free alternating finite automata*

LH . . . . . *logarithmic-time hierarchy* .

LOGSPACE .... *deterministic logarithmic space*

MCS ..... *mildly context sensitive*

MCSG ..... *mildly context-sensitive grammar*

MSOL ..... *monadic second-order logic*

MT ..... *model-theory*

MTS ..... *model-theoretic syntax*

NC ..... *node consistency*

NL ..... *natural language*

NLOGSPACE .. *non-deterministic logarithmic space*

NLP ..... *natural language processing*

NP ..... *nondeterministic polynomial time*

PC ..... *path consistency*

PCG ..... *parallel constraint grammar*

PNF ..... *prenex normal form* – A formula is in a prenex normal form if it consists of a string of quantifiers applied to a quantifier free formula.

POS ..... *part-of-speech*

PSG ..... *phrase-structure grammar*

PSPACE ..... *polynomial space*

PTIME ..... *polynomial time*

Reg ..... *regular language*

RLG ..... *right-linear grammar*

SCG ..... *sequential constraint grammar*

SGP ..... *strong generative power*

SMTA ..... *simple multi-tape automaton*

SRA ..... *specialized rule automaton*

TAG ..... *tree-adjoining grammar*

Type 0 ..... *unrestricted phrase-structure grammars and recursively enumerable languages*

Type 1 . . . . . *context sensitive grammars and languages*

Type 2 . . . . . *context-free grammars and languages*

Type 3 . . . . . *right-linear grammars and regular languages*

WGP . . . . . *weak generative power*

wMSOL$[S]$ . . . . . *weak monadic second-order logic with one successor*

XFST . . . . . *Xerox Finite-State Tool*

XRCE . . . . . *Xerox Research Centre*

Chapter 1

Introduction

Regular expressions (Regexps), *finite automata* (FAs) and other means for expressing regular languages and other regular sets are an attractive research area, although they have an established value in computer science. An interesting area of the research concentrates on methods needed in various applications, and in *natural language processing* (NLP) in particular. In NLP, grammars and related methods based on this kind of *finite-state* (FS) technologies have become very popular and their applications vary from computational phonology and morphology to syntactic parsing and information extraction.

This dissertation belongs to the field of *computational linguistics* (CL), and it studies foundations of FS based grammars and methods that are applicable to syntactic parsing of *natural language*s (NLs). The new results are presented in the accompanying articles that vary in length, formal rigour and field-specific style. Some of them are more convenient for natural language engineers, and some others for mathematical linguists and computer scientists. Nevertheless, I hope that, despite the multiple fields that intersect in this dissertation, this dissertation would be received by people with CLs orientation, and contribute, thus, to our understanding of approaches that FS based grammars provide for syntactic analysis.

In NLP, several approaches to syntactic parsing and disambiguation using FS based grammars have been proposed. In this dissertation, I will be concerned with the theoretical aspects of *finite-state intersection grammar* (FSIG). FSIG is widely perceived as a formally very elegant framework, but it also contains a number of important problems which have remained open for several years. While large-scale FSIG grammars were developed in some earlier FSIG studies, interest in these enterprises faded because of these irritating, foundational problems. In this dissertation, my purpose is to investigate only the foundational problems rather than to build a descriptive grammar or a fully implemented parser.

The Problems in Focus

The general goal of this dissertation is to address the following three closely interrelated problems in the FSIG framework:

1. Complexity Analysis — How much computational complexity is really implied by the languages defined by various kinds of FSIGs? Analyzing complexity of FSIG has earlier been based on a particular automaton implementation of finite-state constraints.

2. Linguistic Applicability — Can some sort of FSIGs represent appropriate structural descriptions for natural languages? Linguistic applicability of FSIG has been restricted to flat surface syntax, where syntactic relations between words are not explicated and transformations that change the order of words are not assumed.

3. Parsing Strategies — How FSIG parsing algorithms could be made more efficient and well-behaving? All FSIG grammars make use of constraints whose representations as *deterministic finite automata* (DFAs) may become rather big. Compact representation of individual constraints and intermediate results in FSIG parsing require new approaches.

My aim has been to gain more understanding of the possible answers to these problems.

The Articles

The dissertation consists of this introductory part and of nine articles that are numbered from 1 to 9. The accompanying articles are:

[1] Yli-Jyrä, 2003a, “Describing Syntax with Star-Free Regular Expressions.”

[2] Yli-Jyrä, submitted 2003, “Regular Approximations through Labeled Bracketing (revised version).”

[3] Yli-Jyrä, 2005a, “Approximating dependency grammars through intersection of regular languages.”

[4] Yli-Jyrä, 2003c, “Multiplanarity – a Model for Dependency Structures in Treebanks.”

[5] Yli-Jyrä and Nykänen, 2004, “A hierarchy of mildly context sensitive dependency grammars.”

[6] Yli-Jyrä, 2004a, “Axiomatization of restricted non-projective dependency trees through finite-state constraints that analyse crossing bracketings.”

[7] Yli-Jyrä and Koskenniemi, 2004, “Compiling contextual restrictions on strings into finite-state automata”

[8] Yli-Jyrä, 2004d, “Simplification of Intermediate Results during Intersection of Multiple Weighted Automata.”

[9] Yli-Jyrä, 1995, “Schematic Finite-State Intersection Parsing.”

The articles are grouped under the three problems. The grouping is illustrated in 1.1 and it shows how the articles included in the dissertation contribute to at least one of these problems.

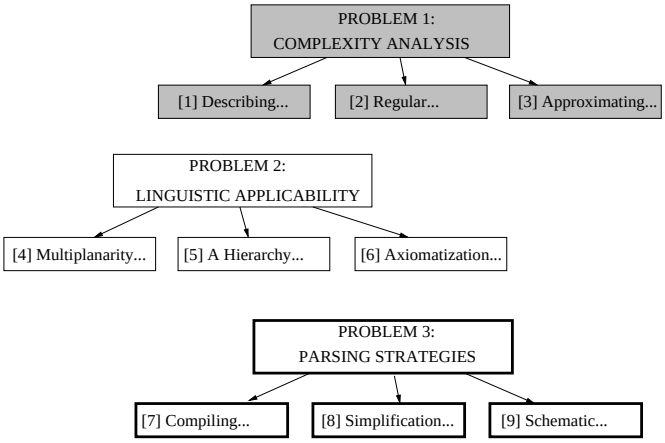


Figure 1.1: The overall structure of the dissertation.

Contributions of this work

This dissertation contributes new insights and a number of new results in relation to the above problems.

Complexity Analysis: Perhaps the most important result in this dissertation is that the languages of structural descriptions defined by many interesting FSIGs [1,2,3,6] are computationally simpler than one would have thought. Some of these grammars are approximations [2,3,6] of new Chomsky-Schützenberger style representations for context-free [2], projective dependency [3] and certain mildly context-sensitive [6] grammars. I analyzed the computational complexity of a specific *English finite-state intersection grammar* (Eng-FSIG) through distinctions provided by the theory of descriptive complexity [1], but it is remarkable that the obtained complexity result extends also to other kinds of FSIG grammars. Furthermore, the study of descriptive complexity (Immerman, 1999) is able to give more structure to complexity analysis when we work on very low complexity classes. Because Eng-FSIG [1] and our new grammars [2,3,6] could be written using star-free regular expressions, their string-based encoding for trees can be defined using first-order logic with linear precedence which means that the computational complexity of these sets belongs to *logarithmic-time hierarchy* (LH).

Linguistic Applicability: The second important result of this work is that FSIGs can be used to give structural descriptions that can be interpreted in terms of fine-grained constituency trees [2], projective dependency trees [3] and interesting classes of general dependency graphs [6]. I carried out [4] some experiments in the Danish Dependency Treebank containing some 5500 sentences. These measurements motivate some performance limits and demonstrate that an appropriate FSIG [6] can cover non-trivial dependency trees in a dependency treebank. Such FSIGs [6] are closely related to some mildly context-sensitive grammars [5] that we presented as a formalization of the complexity measure for crossing dependencies [4].

Efficient Compilation and Representation of Linguistic Generalizations: Finally, this dissertation contributes to the efficient compilation and representation of linguistic constraints by presenting new compilation algorithms [7] and a decomposition technique [7] for FSIGs with bracketing [1,2,3,6]. The relevance of the new, efficient compilation method expands beyond the FSIG framework, to an existing commercial product, *Xerox Finite-State Tool* (XFST). Based on the decomposition technique [7], I sketch an approach [9] that reduces the size of intermediate results [8] during FSIG parsing using an automaton model that resembles incompletely specified asynchronous automata. This approach [8,9] implements a representation for local trees in a FSIG parse forest where trees are represented through bracketed strings.

An Excluded Possibility: The results obtained in relation to the above three problems suggest that algorithms for supervised or unsupervised machine learning of FSIG constraints could be restricted to low dot-depth constraints and to representations that support extended tree locality, and a flexible mixture of dependencies and constituents. The focus of the dissertation excludes intentionally this attractive possibility. Our initial experiments with dependency grammars [3,4,5] touched the learning problem only superficially. In the future, the methods presented in this dissertation could be complemented with techniques for data mining and probabilistic modelling of the data.

The Overview of the Dissertation

The current chapter (chapter 1) started the introduction to the accompanying articles, but a further level of orientational material will be presented in the remaining chapters, characterised as follows: In chapter 2 I give an introduction to the FSIG framework, and relate this framework to some other grammar frameworks and issues in computational linguistics. In chapter 3 I discuss, on the basis of [1-3], complexity analysis and representation of various kinds of FSIGs and characterise our new grammars by a complexity result. In chapter 4 I motivate, on the basis of [4-6], the representation that can encode dependency-based structural descriptions, summarise the related corpus studies and performance restrictions and formalise a class of non-projective dependency grammars and their FSIG approximations. In chapter 5 I point out some problems in efficient FSIG parsing and illustrate how the algorithms presented in [7-9] could help to solve them. The conclusions are presented in chapter 6.

Chapter 2

Introduction to the FSIG Framework

The framework of *finite-state intersection grammar* (FSIG) is a *finite-state* (FS) based approach for syntactic parsing and disambiguation. This chapter presents a general overview of the FSIG framework.

2.1 Background

The basic ideas of the FSIG framework were introduced by Kimmo Koskenniemi (1990). Important contributions to the development of the original FSIG formalism and the first parsing algorithms can be found in a number of fundamental papers (Koskenniemi, 1990, 1997; Tapanainen, 1992, 1997; Koskenniemi et al., 1992; Voutilainen, 1994a, 1997).

Along with FSIG, there are many grammars and efficient parsers whose designs have some aspects in common with it. For example, let us mention *local grammars* (Gross, 1997) and a reductionistic, fixed-point disambiguation approach with context conditions in *sequential constraint grammar* (SCG) (Karlsson, 1990). Because of some similarities between FSIG and SCG, Lauri Karttunen suggested for FSIG the name *parallel constraint grammar* (PCG) in 1994, but the framework got its current, more widely known name FSIG from Piitulainen (1995b)[1]. Set intersection and tree automata for specification of grammars has been employed in studies of descriptive complexity of NL grammars (*e.g.* Rogers, 1998), in the implementation of *local constraints* of Joshi and Levy (1982) and even earlier in computer science (*c.f.* Thatcher, 1967, Peters and Ritchie, 1969).

[1] Piitulainen (1995a) *also* calls FSIG by the name *säännöllinen leikkauskielioppi* that translates to English as *regular intersection grammar*.

2.2 Previous Work

2.2.1 The Original FSIG Grammars

Between 1992 and 1998, Atro Voutilainen developed at the University of Helsinki a large-scale description for English — the Eng-FSIG — using the original FSIG formalism (Koskenniemi et al., 1992; Voutilainen and Tapanainen, 1993; Voutilainen, 1994a,b, 1997, 1998). In addition to this, very similar experimental grammars exist for French (Chanod and Tapanainen, 1994, 1995b, 1996a, 1995a, 1996b, 1999) and Finnish (Heinonen, 1993).

2.2.2 The Original FSIG Representation

The original FSIG parse representation was developed in connection with the Eng-FSIG system. To illustrate the prior art in FSIG, I use this original representation as an example, although it looks in many respects different than the new parse representations that will be briefly referred to in section 2.3.4 and discussed in the subsequent chapters on the basis of my contributed articles.

The Multi-Layered Representation of Parses One of the main innovations in the original FSIG was to combine different annotation layers of the analysis into a unified, one-level representation (Koskenniemi, 1990; Voutilainen, 1994a). This is in contrast to a more popular modular architecture in NLP, where we have *part-of-speech* (POS) disambiguation, clause boundary recognition and syntactic disambiguation in separate modules. The unified representation gives several advantages and opens interesting scenarios for further development. For example, POS disambiguation rules can now have access to the syntactic functions of words, and clause boundaries can be used to disambiguate these syntactic functions, or alternatively, the dependencies can also work in the opposite directions.

An Example of the Unified Representation When all different annotation layers of the analysis are combined we get a rich representation of the analysis as an annotated surface string. A portion of such a string is shown in the following (I wrapped the string on multiple lines and added the tabbing):

```
                                                    @@
in      <*>             PREP                    @ADVL     @
the     <Def>           DET CENTRAL ART SG/PL   @>N       @
1950s   <1900s>         NUM CARD                @P<<      @<
                                                @comma    @
as                      CS                      @CS       @
african <*> <Nominal>   A ABS                    @>N       @
nation                  N NOM PL                @SUBJ     @
prepare <SVOO> <P/for>  V PAST VFIN             @MV ADVL@ @
for                     PREP                    @ADVL     @
independence <-Indef>   N NOM SG                @P<<      @
                                                @comma    @>
...                     ...                     ...       ...
```

An Illustration of Different Annotation Layers If we want, we can also view this unified representation through different projections where features irrelevant to a particular layer are hidden. For example, we can view the clause structure, in which case I may get the following string:

```
@@ In the 1950s @< , as African nations prepared ADVL@
for independence, @> these highly charged poetic
images @< , which emphasized N<@ the humanity of black
peoples, @> gave MAINC@ way to prose @/ that
satirized N<@ the colonizer and the colonized.  @@
```

In this string, the tags @@, @/, @< and @> denote, respectively, sentence boundaries, iterative clause boundaries, and embedding clause boundaries. The syntactic function of each clause is shown with tags ADVL@, N<@, and MAINC@, that indicate, respectively an adverbial clause, an attributive clause and the main clause.

Another view would show the main syntactic functions in each clause:

```
In @ADVL the 1950s, as @CS African nations @SUBJ
prepared @MV for @ADVL independence, these highly
charged poetic images @SUBJ, which @SUBJ
emphasized @MV the humanity @OBJ of black peoples,
gave @MV way @OBJ to prose that satirized @SUBJ
the colonizer @OBJ and the colonized @OBJ.
```

In this view we see adverbials (@ADVL), subordinate conjunction (@CS), subjects (@SUBJ), main verbs (@MV), and objects (@OBJ). We could also have a look at modifiers of nouns (@>N, @N<, N<@), and complements of prepositions (@P<<), in which case we get the following view:

```
In the@>N 1950s@P<<, as African@>N nations prepared
for independence@P<<, these@>N highly@>A charged@>N
poetic@>N images, which emphasized N<@ the@>N humanity
of@N< black@>N peoples@P<<, gave way to@N<
prose@P<< that satirized N<@ the@>N colonizer and
the@>N colonized.
```

Morphological analysis of each word is given using a lemma and a fairly standard morphological tags such as N, DET, A, ADV *etc*[2]. This information and *e.g.* subcategories of the verbs, such as (<SVO>, <P/for>, <P/with>) is indicated right after each lemma.

2.2.3 Parsing Algorithms

The first parsing algorithms for FSIG have been developed by Pasi Tapanainen who worked at the University of Helsinki in Finland (Koskenniemi et al., 1992; Tapanainen, 1991, 1993, 1997, 1999), and at the *Xerox Research Centre* (XRCE) in Grenoble, France (Chanod and Tapanainen, 1994, 1995b, 1996a, 1995a, 1996b, 1999).

[2]Both the lemma and the surface form could have been indicated (see Voutilainen, 1994a). The surface form is not necessarily identical to the phonological form, but means any kind of string that is used to interface the syntax to the morpho-phonological component.

Besides Tapanainen, alternative approaches to FSIG parsing have been proposed by several other authors (Piitulainen, 1995b; Voutilainen, 1998; Lager, 1999; Lager and Nivre, 2001; Yli-Jyrä, 1995, 1997, 2001, 2004d) but most of these parsing methods are poorly tested[3]. Using FSIG as a preprocessor for deep-syntactic parsers has been investigated by Lounela (1993).

2.3 Some Design Aspects of the Framework

2.3.1 Generative or Non-Generative Grammar?

I suspect that many non-mathematically oriented linguists who do not like generative linguistics tend to think automatically that every *formal grammar* (FG), such as FSIG, is based on the generative framework. Because I would like to motivate us to see how much the basis of FSIG really differs from the *generative-enumerative syntax* (GES) approach, I have to explain this distinction.

Generative-Enumerative Framework The notion of a FG was introduced by Chomsky (1957). In particular, Chomsky introduced *phrase-structure grammar* (PSG) as a formal device that enumerates the sentences of a language one after another. This kind of grammar generates (derives, produces) the sentences in the language and only those. These PSGs use so-called production rules and they have their formal basis in Post's inference rules (Post, 1943). Post's inference rules arrange and rewrite strings of symbols, which serves as a combinatorial foundation for proof theory – the syntactic side of logic. The approach where grammars are based on such syntactic inferences will be called here GES. (*C.f.* Pullum and Scholz, 2003.)

When GES grammars are used for NL description, they take an *extensional* view to NL: they identify the described NL with a unique set of strings. Utterances that do not belong to the identified set are not considered to be members of the language being described. Kornai (1985) points out that the same idea was implicit in much of the work of the structuralist period, in writings of Bloomfield and Harris for example. Also some early computational linguists, including Abraham (1965), have claimed that the only reasonable way to define infinite sets of utterances is a GES grammar.

The Chomsky Hierarchy A standard and very useful measure of the complexity of *formal language*s (FLs) is provided by the *Chomsky hierarchy* (CH) (Chomsky, 1956). CH consists of the following four classes of grammars and languages: *unrestricted phrase-structure grammars and recursively enumerable languages* (Type 0), *context sensitive grammars and languages* (Type 1), *context-free grammars and languages* (Type 2), and *right-linear grammars and regular languages* (Type 3). All classes contain infinite languages and their grammars, but the largest class is Type 0 and its capacity corresponds to Turing machines. Determining the smallest class containing a language described by a grammar is often difficult. The Type 3 languages

[3]Some of the methods that were presented by Yli-Jyrä (1995, 1997) have been tested in practice, but there is no publication on these experiments. Furthermore, Voutilainen (1998) combines SCG and FSIG in thorough experiments.

correspond to *finite automata* (FAs). Finite-state models have been used to model the capabilities of language users (Miller and Chomsky, 1963).

Many aspects of FSIGs can be approached through formal notions of FGs and CH. For example, the idea of finite memory such as in FAs is built in FSIG, and the capacity of FAs correspond to the Type 3 languages.

FSIG Is Not a GES Framework FSIG constraints can potentially express any regular set over the given alphabet, being thus similar to *right-linear grammar*s (RLGs). However, there are several reasons why FSIG as a whole is not based on the GES approach, or that it is at least ambiguous:

- It does not directly define phrase structures,
- It does not use productions, and
- It does not derive the sentences with syntactic inferences.

FSIG Is Based on Model-Theory Instead of having productions, FSIG contains assertions. The basis of FSIGs is, thus, in set-theory and *model-theory* (MT) – the semantics of mathematical logic. MT studies the relationship between logical assertions and the interpretations that are defined for them. An interpretation of an assertion (or a corresponding set of objects) is given by saying whether a given structure satisfies it. A structure that satisfies the assertion is one of its models. This is very similar to the one-level view of PCG as presented by Koskenniemi (1997), where the input is classified by the yes/no answer by syntactic assertions.

Some basic ideas of model-theoretic frameworks were already presented in the 1960's by Elgot (1961) and Thatcher (1967) and later by many others, especially those working *e.g.* on constraint-based linguistic formalisms. A recent way to define the framework of *model-theoretic syntax* (MTS), and its linguistic aspects in particular, has been advocated for instance by Pullum and Scholz (2003)[4]. All MTS frameworks are sets of assertions about the syntactic properties of linguistic utterances. I do not see any reason why FSIG could not be recognized as a particular system in the framework of MTS.

2.3.2 The Assertions

Universal and Parochial Constraints In FSIG, all assertions are usually called constraints or (constraint) rules. The first subset of these constraints contains *administrative constraints* or *axioms* that define general structural properties (encodings, treeness, rootedness, *etc*.) of the domain of models we are restricting with other assertions. These constraints should not not differ from language to language, because they form syntactic theories that could be associated with particular types of FSIG. Beside the axioms, there are assertions that are true only for particular languages. These assertion could be called *parochial constraints* (Pullum and Scholz, 2003).

[4]There must be better references to MTS, such as (Miller, 2000), but this is the one I am acquainted with. Unfortunately, the availability of this ESSLLI reader (Pullum and Scholz, 2003) is not the best possible.

Example For instance, we can mention the following axioms that contribute to the specification of the possible parses in Eng-FSIG:

- All the analyses begin and end with tag @@.
- If a string contains a tag @< it is followed by at least one tag @> in the string.
- At any position of the strings, the number of open @< brackets does not exceed a given limit.

The Logical Formalisms In MTS the assertions are normally expressed in a logical formalism. However, Büchi (1960) showed that regular expressions are equivalent to *weak monadic second-order logic with one successor* (wMSOL$[S]$) interpreted over finite strings. Due to this result, regular expressions can be seen as a notational variant for wMSOL$[S]$.

The constraints in FSIG grammars are normally expressed using special FSIG regular expressions, which contain some extensions. The extensions make them more convenient for expressing some constraints. The most important extension is the so-called *context restriction operation* (arrow rule, implication rule) (Yli-Jyrä and Koskenniemi, 2004) that has always been part of the FSIG framework (Koskenniemi, 1990).

None of these two formalisms directly make a reference to automata or production systems. However, regular expressions can be compiled into automata or *e.g.* interpreted lazily as grammars. Similarly, wMSOL$[S]$ can be interpreted lazily using a recursive function or implemented *e.g.* by compiling the formulas into automata.

Normally, the structures that satisfy wMSOL$[S]$ formulas are called *models*. However, strings being described by a regular expression are not normally called models, although it is quite common to say that such strings match a regular expression, or satisfy a constraint. When we refer to a string that satisfies a constraint and, equivalently, belongs to a constraint language, I will call such a string a *model string*[5].

By a constraint language in FSIG, I will refer to the set of all string that satisfy the given constraint. Thus constraint languages do not refer to the logical *etc*. formalisms that are used for expressing constraints.

The Essentials of the Constraint Formalism Some examples of possible uses of context restrictions are in figure 2.1. The first implication rule says that the leftmost auxiliary verb (`@AUX`) must be a finite verb (`VFIN`). The second rule illustrates the fact that the left-hand side can be a rather complicated regular expressions. The third rule illustrates the fact that the context conditions on the right hand-side can be multiple, bilateral and rather complicated. Special symbols such as `'.'`, `'..'`, `'>..<'` and `'..not/1'` in these rules denote specially defined regular languages and macros that make the formalism slightly easier to use.

In addition to context restrictions, various FSIG formalisms have two important features in common: (i) special variables (during the grammar compilation they are fixed constants) provide an easy way to make reference to balanced bracketing. (ii)

[5]This is perhaps not the best term, but terms such as *recognized string* and *matched string*, *parse string*, *annotated string* would not be any better.

```
                                @AUX  =>  VFIN . _ , @AUX .. _
    @MV SUBJ@ >..< @> ..  @MV MAINC@  =>  <SUBJ> >..< _
                                @oc   =>
OC . @mv >..< @obj ..not(@AUX | @MV)  _  ,
          make .  @mv .  @ .  Adj .   _  ,
                              Adj .   _  [@ .  [@>A|@A<]]* @ .  @obj ,
                               WH .   _  .  @ to .  @ .  OC . @mv >..< @obj
```

Table 2.1: Some example rules taken from Eng-FSIG.

left- or right tail recursion in nested bracketing is eliminated by using an appropriate bracketing schema.

2.3.3 The Sets Defined by Grammars

Grammatical Model String Sets Defined by FSIGs The set of *grammatical* model strings for an FSIG grammar consists of those strings that satisfy all individual constraints in the grammar. Obviously this set of strings is equal to the intersection of all constraint languages in the grammar. In the case of the Eng-FSIG, the intersection has the structure

$$\underbrace{B(WFB)^+}_{\text{“domain”}} \cap \underbrace{C_1 \cap C_2 \cdots \cap C_{n_k}}_{\text{local lexical constraints}} \cap \underbrace{C_{n_k+1} \cap C_{n_k+2} \cdots \cap C_{n_c}}_{\text{other constraints}},$$

where B is an *alphabet of delimiters* (boundaries), W is a finite *set of morphological analyses*, F is a finite *set of syntactic functions*, and $C_1, C_2, \ldots C_{n_c}$ are *sets of model strings* defined by different constraints. The *local lexical constraints* are those whose function is to relate word forms with their possible syntactic functions (Yli-Jyrä, 2003a).

Intersection of Constraint Languages Is Effective It is a classical result that regular languages are closed under intersection. Because FSIG constraint are regular expressions they describe regular expressions and their intersection will be *regular language*s (Regs) as well. One way to compute the intersection of these Regs is to compile the regular expressions first into FAs, then to compute the direct product of these automata (Rabin and Scott, 1959). The direct product of the automata is an automaton that recognizes the intersection of the languages described by the corresponding regular expressions. Although this is formally an elegant definition, its practical implementation is quite complicated.

Gradient Ill-Formedness An over-constrained FSIG grammar defines an empty set and is, thus, uninteresting as a GES grammar. Pullum and Scholz (2003, p.59,60) argue, however, that the MTS framework offers an elegant starting point for thinking about the comparative ungrammaticality of ill-formed sentences. Thus, it is not a task of MTS grammars to define the set of grammatical model strings and only those. In fact, over-constrained grammars may still be useful since they define gradients of ill-formedness (if constraints are turned on and off) and they could be used for robust parsing. This is one of the most important differences between GES and MTS

grammars (optimality theoretic grammars in particular). In fact, Voutilainen (1994a) already included some heuristic constraints as parts of the Eng-FSIG system. Weighted constraints can reduce the ambiguity that remains after all hard constraints have been applied in parsing.

In order to keep my current investigations within reasonable complexity, I do not consider gradient ill-formedness nor parse ranking. Nevertheless, such notions might be necessary for robust parsing (and for text parsing) where it is important to find the best candidate analyses. In this dissertation, the language of each FSIG will be uniquely the set of grammatical model strings, which makes FSIG superficially comparable to GES grammars.

2.3.4 String Encoding of Syntactic Entities

FSIG encodes all the parses as strings rather than as trees. This is an important feature that differentiate FSIG from approaches using tree-automata (Joshi and Levy, 1982; Rogers, 1998). Strings and string sets (and regular expressions and automata) have been used both inside most FSIG parsers and in their application interfaces. Internal and external representations are not necessarily identical.

External Representation of Parses In section 2.2.2, I already described the original representation of parses used in the original FSIG. This representation is in many respects under-specified and it avoids many difficult syntactic decisions concerning the linking of words. Such under-specification can be useful in shallow and partial parsing methods and provide a compact representation for unresolved ambiguity (*c.f.* Church and Patil, 1982).

In a number of applications, we would prefer *full* syntactic structures as analyses produced by the parser. Fully specified syntactic structures of various sorts (constituent trees, dependency trees) can also be encoded as strings. In this dissertation, a few such encodings are proposed. At this point, I merely make reference to the accompanying articles (Yli-Jyrä, submitted 2003, 2005a, 2004a) that will be discussed in the subsequent chapters of this dissertation. Sometimes these encodings become so complicated that their direct inspection by the grammar developed is no more very convenient and decoding to trees and other more user-friendly visualizations is needed.

Internal Representation of Parses Classical finite-state automata are useful in representing sets of parses inside FSIG parsers. Sometimes they are, however, compact enough for practical implementation. In chapter 5, I will discuss methods that are motivated by the optimization of the automaton based implementations. Some of the methods change the representation of parses as strings by introducing new details into the letters in the strings. The resulting representations allow simplifications to the automata in a surprising way: individual strings become more complex but the added details make the string sets simpler and thus easier to represent compactly as parallel decompositions of automata. This technique makes use of the fact that the regular languages are closed under string homomorphisms (Hopcroft and Ullman, 1969).

2.3.5 Layers, Planes and Projections

The idea of multiple layers in the original FSIG can be brought further. We can define and use layers in many different ways:

1. When verifying that bracketing used in the annotation is balanced as unlabeled bracketing, we can project labeled brackets to unlabeled ones (Yli-Jyrä, 2005a).

2. In order to implement a bracketing scheme for *colored non-projective dependency grammar*s (CNDGs) I separated "planes" that correspond to different colors of the dependency links (Yli-Jyrä, 2004a).

3. As suggested in the end of one of my articles (Yli-Jyrä, 2004a), several grammars can be used in parallel (*e.g.* a dependency grammar and a grammar for topological fields) if they share the same one-level representation, but use partly different layers (or tapes).

4. In order to optimize the automata that define constraint languages, I sometimes need methods that can decompose constraints in an FSIG into a number of sub-constraints (Yli-Jyrä and Koskenniemi, 2004). According to my proposal, the sub-constraints see everything in the one-level representation but they define generalizations of the original constraint and accept more.

5. The alphabet in the encoding can be extended so that it becomes possible to extract a layer from the extended one-level representation and to express a sub-constraint even without seeing the whole string. Alphabet extensions can be bound to a measure of parameterized structural complexity, such as number of nested brackets (Yli-Jyrä, 2004d), or it can be done on the sentence-by-sentence basis (Yli-Jyrä, 1995).

2.3.6 The Generative Capacity – How to Characterize It

The notion of *strong generative power* (SGP) of a grammar formalism is often used to talk about appropriateness of a grammar. One approach to the question of appropriate models for syntactic structures is considering how much SGP is really necessary when certain kinds of structures are being described.

The Basic Definition for GES Grammars The classical notion of generative capacity — *weak generative power* (WGP) and SGP — were introduced by Chomsky (1963) in the context of the theory of formal grammars and automata.

- The WGP of a GES grammar is defined as the language generated by the grammar. For other formal grammars, the WGP of a grammar is defined as the set of strings defined by the grammar.

- The SGP of a GES grammar is defined as the set of structural descriptions that the grammar assigns to the strings it generates.

These notions generalize to grammar theories or classes of grammars. The WGP and SGP of a GES theory are respectively the set of WGPs and the set of SGPs of the grammars that are encompassed under that theory or class of grammars.

Defining SGP of MTS Grammars In order to generalize the notion of SGP to cover both GES and MTS grammars, Miller (2000) proposes that SGP should be understood as the model-theoretic semantics for a linguistic formalisms or syntactic theories. Before defining SGP of all these syntactic theories, we need to define abstract interpretation domains defined for such common notions as labeled constituency, dependency, endo-centricity, and linking. This must be done in a theory neutral way, using *e.g.* set theoretical terms. With respect to such an interpretation domain, the SGP of a class of grammars is characterized as a range of a specific interpretation function that maps structural descriptions of that class of grammars to the elements of this interpretation domain[6].

Defining SGP of FSIGs The SGP of an FSIG is obtained from the set of structures that are obtained by interpreting the set of grammatical model strings. Different types of FSIG use different representations, and their SGP is obtained by choosing an interpretation function that is defined appropriately for each type.

2.3.7 Parsing with FSIG

The Proper FSIG Parser and Other Requisites FSIG can be used for parsing. Preparing an input, *surface string* for application of FSIG constraints, is a process that is carried out by using possibly a (tokenizer-)*morphological analyzer* and *lexical lookup or inverse homomorphism mapping* in such a way that we obtain a set of potential *deep strings*. Then FSIG constraints are used for selection of grammatical model strings out of the set of deep strings. Although the whole process is related to FSIG parsing, the *proper FSIG parser* consists only of the lexical lookup or inverse homomorphism and the set of FSIG constraints. This is illustrated in figure 2.1.

Reduced and Holistic Definitions for FSIG (Koskenniemi, 1990) distinguishes two views on PCG *i.e.* FSIG: according to the *one-level*, FSIG, being a combination of constraints, is also itself a complex constraint. It works like an acceptor, telling us whether a deep string satisfies the grammar or not. Although its output is really a binary yes-or-no answer, the "yes" answer will indicate that the deep string is a grammatical model string. Therefore I have depicted the combination of FSIG constraints as a relation between all deep strings and those deep strings that are grammatical model strings. According to the *two-level view* of FSIG, I consider the proper FSIG parser as the FSIG. I can also define *a holistic view* according to which any process can be called an FSIG parsing system if its output is an output of a proper FSIG parser that is embedded into it. The output that consists of the deep strings that satisfy the grammar are practically called *parses* of the surface string.

[6]This terminology originates from the notions in denotational semantics of logical formalisms.

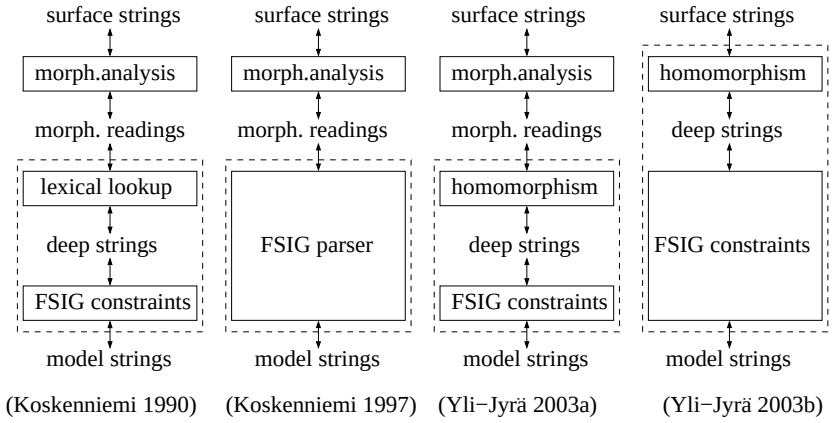


Figure 2.1: Alternative ways to implement parsers for FSIG.

In this dissertation, I will concentrate on the one-level view of FSIG (*c.f.* Yli-Jyrä, submitted 2003, 2005a, 2004a).

Sentence Automaton For each surface string, its all deep strings *i.e.* the set of potential model strings, will be called according to Piitulainen (1995a) as the *initial set* (*alkujoukko* in Finnish), and it is usually represented by a *deterministic finite automaton* (DFA) that is called a *sentence automaton* (in fact the initial version of it).

According to Tapanainen (1997), the constraints can be *applied* to a version of the sentence automaton. This means that a constraint language, represented as an DFA, is intersected with the language recognized by the sentence automaton by computing a new automaton that is similar to the direct product of the two automata except that it can be minimized. This new automaton becomes then the next version of the sentence automaton. During the parsing process, we may obtain, thus, a series of versions of the sentence automaton so that the number or deep strings recognized by them monotonically decreases.

Specialized Grammar and SRAs In the approach of Piitulainen (1995b), the constraint automata and the initial sentence automaton are applied during the parsing process to construct a special data structure. In Yli-Jyrä (1995) this data structure is represented distributively by *specialized rule automata* (SRAs) whose collection is called a specialized grammar. Specialized grammar can be seen as a compact representation for ambiguity although it may require further processing and simplifications before it reaches the point where it is efficient to consult it for finding grammatical model strings by a lazy intersection algorithm.

2.4 Relevance of FS Based Grammars

2.4.1 Can FS Grammars Be Linguistically Motivated?

Practical Point of View It is a common belief that finite-state grammars are linguistically inadequate for natural languages. Due to this widely accepted standpoint, there is a tension between finite-state based syntax and non-finite-state frameworks such as Transformational Grammar (TG), Categorial Grammars (CG), Tree-Adjoining Grammars (TAG) and Lexical Functional Grammar (LFG).

I would like to see that this tension is mostly a practical question, and I would probably agree with many about the advantages of non-finite-state formalisms. In my dissertation, we actually see that non-finite-state frameworks such as *context-free gramma*rs (CFGs), dependency grammars and mildly context-sensitive grammars are very useful for modeling tree locality and co-occurrence constraints in linguistic structures, but finite-state grammars, and FSIG in particular, are at their best when used to approximate computationally expensive formalisms under linguistically appropriate performance limits. Such limits are well-motivated when the language is being processed by humans as well as by computers.

The Myth about Non-Regular Natural Language A less practical, but theoretically interesting question is whether natural languages, as string sets, could be modeled by a finite-state mechanism. Adequateness or inadequateness of finite-state or regular grammars for modeling the surface of natural language is seldom discussed critically in the contemporary linguistic literature, and some may be misled to think that it has been been proven that natural languages are not regular languages. However, some decades ago there was a time when inadequateness of finite-state grammars was not taken for granted by linguists. Arguments against the adequacy of finite-state grammars were presented by several authors, most notably by Chomsky (1957) and Bar-Hillel and Shamir (1964).

I will now rewind back to that discussion and recall some important reasons due to which the adequateness problem has actually remained an open, perhaps unsolvable problem.

Some Problems in Non-Regularity Arguments Chomsky (1959b,a) and Bar-Hillel et al. (1964) showed that arbitrarily deep self-embedded structures require unbounded memory in general. However, even if the self-embedding[7] of a context-free grammar is not bounded, the language of the grammar can still be regular. In fact, the problem of regularity is, in that case, undecidable (Ullian, 1967)[8]. This means that showing that a grammar is self-embedding does not prove that the language is not regular.

In formal language theory, a proof method that makes use of the so-called pumping lemma is useful in showing that certain languages are not regular. However, this method is particularly difficult to apply validly to natural languages.

[7] Phrase A is a *self-embedding* if it is embedded into another phrase B of the same category and surrounded by lexical material in B.

[8] The problem becomes decidable when the context-free grammar is deterministic (Stearns, 1967) or strongly regular (Nederhof, 2000).

Chomsky's original attempt to demonstrate that FS models are inadequate (Chomsky, 1957, ch. 3.1) was based, first, on a crucial assumption that NLs are self-embedding to an arbitrary degree, and, second, on a fallacious assumption that if a subset of a natural language is not regular then the whole natural language is not regular. A better argument can be based on the first assumption and the fact that regular languages are closed under intersection: if the intersection of the natural language and a regular language is not regular, then the natural language in question is not regular either. This improvement have been proposed by Brandt Corstius (see Pullum 1984). Chomsky's original argument (1957) is problematic also because it assumes that the best generalization of the linguistic examples is among those grammars that are brought into the discourse (*c.f.* the critique Pullum and Gazdar, 1982; Wintner, 2001 related to certain similar arguments that have been presented against context-freeness of natural languages).

The Crucial Assumption In the above, the crucial assumption was that NLs are self-embedding to an arbitrary degree. This assumption involves subtle problems. In the following, I summarize some of the possible approaches.

- Because the self-embedding problem is connected to context-freeness, the conclusions will be dependent on the types of PSGs available in the Chomsky hierarchy. Accordingly, Joshi et al. (2000) notes that only performance constraints can limit self-embedding depth, but he claims, at the same time, that limited scrambling complexity can be captured by competence differences (different classes of mildly context-sensitive grammars). There might be ways to see this kind of performance-competence division as an artefact of the hierarchy induced by the types of PSGs.

- In order to have unbounded self-embedding, I have to assume that the natural language is infinite, *i.e.* there is no upper bound for the length of the strings in the language. While Pullum and Scholz (2003) argue that many classical arguments for infinitude of natural language are circular or otherwise without independent support, statistical (Zipfian) models (Kornai, 1999, 2002) seem to provide arguments according to which languages – at least their lexicon – is infinite.

- On one hand, it has been noted that the data of certain non-European languages of the world demonstrates that several clauses can be systematically be inserted (self-embedded) in the middle of each other (Pullum, 1984). On the other, corpus-based studies and psycholinguistic studies demonstrate that there is a performance degradation in the case of deep embedding.

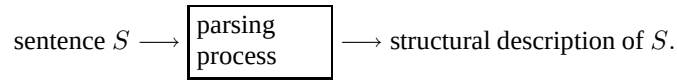
- Some "magic" limits for the depth of self-embedding has been proposed (De Roeck et al., 1982; Lewis, 1996). A better grounded hypothesis is presented in a recent corpus study (Karlsson, 2004 (in print)) that proposes that the categorial restrictions on clausal embedding become tighter when embedding becomes deeper. Such restrictions would mean that self-embedding is not allowed at all.

It seems to me that the assumption of the existence of self-embedding in competence remains as a possible hypothesis. What makes us anxious is that the hypothesis may even remain unfalsifiable. We need, thus, a more practical point of view.

2.4.2 Importance of Finite-State Based Grammars

The Principle of Scientific Parsimony In science, the most easily falsifiable but not yet falsified theory is preferred to theories that are hard to falsify (Popper, 1959). As pointed out by Kornai (1985), the principle of scientific parsimony suggests the minimal language family suitable for the defining natural language string sets. For this purpose, regular languages provides a generous upper bound, because they contain every finite language and they are closed under regular operations.

Efficient Parseability Ejerhed and Church (1983) point out very illustratively that in NLP the motivation for finding restrictions to the theory of syntax is quite different from Chomsky's motivation, as he uses the universal constraints to give an explanation for grammar acquisition process. In natural language and speech processing, I aim at efficient implementation of the parsing process:



The need to make parsing more efficient motivates restrictions that make the theory of syntax less powerful and the grammars easier to process.

Ejerhed and Church (1983) are convinced that the explanations for parsing speed and acquisition speed are related, and not separate and unrelated. Rather than viewing limitations as something that does not belong to linguistic competence, we can, if we want to, consider limitations as constituting the very essence of an explanation for why languages are structured the way they are.

In order for linguistic theory to have psychological reality it is necessary to build a variety of performance limitations, like memory limitations, into the grammar. Hale and Smolensky (2001) have shown that a simple performance theory can be constructed that incorporates a competence theory in a relatively straightforward way into a procedural specification for parsing. Still, there is no logical necessity that performance and competence will ultimately converge in every area (Church, 1980).

2.5 The Main Open Problems in FSIG

2.5.1 Computational Complexity and Parsing Strategy

The most important open problem in the FSIG framework is its parsing inefficiency. When a parser for Eng-FSIG was implemented using an automaton synthesis approach, the computation took an unbearable amount of time (Koskenniemi et al., 1992, Tapanainen 1991, 1993, 1997). In this approach the product automaton is build from the sentence automaton and the automata recognizing individual constraint languages.

According to Tapanainen (1997), a *depth-first search* (DFS) algorithm, which has exponential-time complexity (according to the sentence length), is in practice much faster than a number of linear-time construction algorithms which he also investigated from many different perspectives. The DFS algorithm is more commonly known as the backtracking search algorithm. The backtracking search is, however, prone to unexpectability and it could not guarantee, for example, that any 35-word sentences would have been parsed in a reasonable time with the current Eng-FSIG that contains 2600-assertions. For example, in 1998 Tapanainen's simplified DFS-based parser that was made available to me required roughly one week to complete parsing of a 35-word sentence. In 1997-1998, I implemented a parser that was basically based on an improvement of Tapanainen's search algorithm. It needed 9 hours to do the same task on the same machine.

Several alternative approached to the parsing problem have been proposed (Piitulainen, 1995b; Voutilainen, 1998; Lager, 1999; Lager and Nivre, 2001; Yli-Jyrä, 1995, 1997, 2001). However, none of these seem to provide a sufficient strategy for solving the whole parsing problem efficiently.

The Ground for Optimism Are there any grounds for being optimistic with respect to the complexity of FSIG?

Yes, we can be optimistic. The prior experiments have given the following encouraging observations:

- Sometimes computing intersection of languages of hundreds or thousands of finite automata can be done within a reasonably short time (Koskenniemi et al., 1992). If this had not been possible at all, it had been impossible or at least very difficult to develop Eng-FSIG, which now contains as many as 2600 rules.

- Although parsers can construct large automata during the application of the grammar rules, their final result – if the parsing can be completed – is a very small automaton.

The second observation provokes an optimistic dream: the structure of English and the analyses of Eng-FSIG perhaps imply a favourable property that guarantees a small number of parses for each sentence[9]. It has been an open problem whether this suggested nice property could be explained with a better asymptotic bound for the computational complexity of the grammar or parser.

In this dissertation I will investigate the computational complexity of the parse language of FSIGs and how structural properties of this set can be made use of during FSIG parsing.

2.5.2 Linguistic Adequacy

The second important open problem in the FSIG framework is the question of its linguistic applicability. It has been thought that FSIG is suitable only for tagging and

[9]The PP attachment ambiguity was never a problem in Eng-FSIG because of its structural annotation was under specified.

shallow syntax. By shallow syntax one usually means: (i) that the grammar is so loose that it cannot be used for specification of a set of grammatical strings (in the GES sense), or (ii) that a grammar does not describe long distance dependencies, crossing dependencies *etc*. complex phenomena, or (iii) that the representation of the parses are build upon the surface string by means of functional annotation, bracketing, and traces.

It is true that FSIG does not define a deep representation layer where the word order of the sentence would be in a canonical form. However, there are certainly other reasons to think that FSIG can give adequate structural descriptions to the sentences.

As to the linguistic adequacy of FSIG, this dissertation will investigate what kind of descriptions can be obtained by means of new kinds of FSIGs.

2.5.3 Other Open Problems

In connection to FSIG, there remain many interesting questions that cannot be addressed here. For example, I would like to investigate how FSIG grammars could be learned by a machine from a corpus or a treebank. Machine learnability of natural languages is an important and very popular research subject. I believe that although the current work leaves the question almost intact, it conveys such insight on locality in FSIG grammars that may be valuable when we approach the machine learnability of FS based NL grammars.

Chapter 3

Complexity Analysis

3.1 Orientation

3.1.1 Problem

This chapter investigates the computational complexity of checking whether a deep string (annotated surface string) satisfies FSIG constraints. Checking a property of deep strings is often easier than parsing from surface strings. However, even easy problems differ in complexity and such differences are useful because they can suggest new parsing strategies.

My approach to investigate the computational complexity of grammars differs from the usual approaches. The computational complexity of constraint languages will be measured here in the light of descriptive complexity, extended star height, and the *dot-depth hierarchy* (DDH) of star-free regular languages.

3.1.2 Articles

The investigations will cover, basically, the following three different FSIGs:

- one-level Eng-FSIG [1],
- regular approximations of *flat context-free bracketing grammar* (FCFBG) [2]
- regular approximations of *Hays-Gaifman dependency grammar*s (HGDGs) [3]

These grammars have been investigated in three accompanying articles, in the following way:

Eng-FSIG

> [1] Yli-Jyrä, 2003a, “Describing Syntax with Star-Free Regular Expressions”

The article is the first in a series of articles on a *new family* of finite state intersection grammars. All the grammars in these articles have a special relationship with star-free regular expressions.

In this article, I employ *heuristic techniques* that can be used for reducing regular expressions or extended regular expressions into extended regular expressions whose star-height is zero. On the basis of these techniques, I can argue that all FSIGs in this dissertation (with some reservations on FCFBGs) are expressible without stars, although I prove this explicitly only for Eng-FSIG. Due to the connection between extended star-freeness, first-order logic and descriptive complexity, the result implies an *improved upper* bound for the parallel time complexity of recognition of deep string languages that are expressed by star-free grammars.

In addition to these results, the article also gives some useful insight into *limited balanced bracketing* (LBB) and into the *context restriction operation* that is a useful extension to regular expressions in the original FSIGs. The subsequent articles in this dissertation develop these themes further.

FCFBG

> [2] Yli-Jyrä, submitted 2003, "Regular Approximations through Labeled Bracketing (revised version)"[1]

In the article, I relate production schemata of extended CFGs to *bracketing restrictions* that are assertions in FCFBGs. The resulting grammars can be approximated with FSIGs by restricting the depth of bracketing in these grammars. The article proposes a measure for the complexity of the resulting approximations. The measure is based on the DDH of star-free languages (Cohen and Brzozowski, 1971). The article also discusses *deterministic state complexity* of bracketing restrictions and whole grammars.

Besides these complexity results, the article contributes a bracketing scheme, called *reduced bracketing*, that can be employed to retain linguistically interesting coverage of tail recursion in approximations, and proposes a non-standard, intersection based compact representation for regular approximations of context-free bracketing grammars.

HGDG

> [3] Yli-Jyrä, 2005a, "Approximating dependency grammars through intersection of regular languages"

In this article, I develop a novel string-based *encoding for dependency trees* and a unique Chomsky-Schützenberger style *grammar representation* for HGDGs. From such a grammar representations I obtain a special class of FSIGs by limiting the nested brackets. Dependency parsing with the obtained FSIG approximations has *linear time complexity*. In these FSIGs, most constraints are combinatorially very cheap because

[1] This article is a heavily revised version of an article (Yli-Jyrä, 2003d) that was presented in *Formal Grammar 2003*. The proceedings of the revised conference papers were still under construction when this dissertation was printed.

their combination is reduced to their *strictly locally testable properties*[2] rather than cross-product of their state spaces. The remaining constraints check that the bracketing in strings is well formed. This illustrates a very important point that the hardest part in FSIG parsing is to match brackets with each other. Finally, I argue that there is some hope for overcoming the state space explosion of intermediate results. The paper reports tiny parsing experiments with some grammars that were extracted from a corpus. (Due to space limitations, the paper does not study grammar extraction as a problem *per se*.)

Besides these results and proposals, the article comments on axiomatisations and properties of planar and projective dependency graphs as well as acyclic projective dependency graphs [3].

3.1.3 Outline

I will introduce the three articles in sections 3.2 - 3.4. Before that, I include in the following a short summary of previous results on computational complexity of FSIG parsing. The whole chapter is closed by section 3.5.

3.1.4 Previous Work

Computational complexity measures the amount of computational resources (such as time, space, parallelism and random bits, *c.f.* van Melkebeek, 2000) that are needed, as a function of the size of input, to compute an answer to a query.

The Inclusion of FSIG Grammars into the Linear Time Complexity Class It is well known that a direct product of two deterministic finite automata – and the intersection of their languages – is computable in a linear time according to the sizes of the automata (Rabin and Scott, 1959). During the FSIG parsing, the set of constraints in the grammar is fixed and the intersection of the constraint languages can be implemented by an automaton that does not change. The direct product of this constant automaton with the initial sentence automaton can also be computed in linear time according to the length of the sentence (Tapanainen, 1997) [4].

One can arrive at the same conclusion by thinking about the regularity of FSIG. Because each fixed FSIGs describes a regular language, we know that there is some non-deterministic or deterministic automaton accepting it and it can, thus, be accepted in linear time. Combining finite-state morphological analyzer, lexical lookup and a

[2] Actually, many constraints are not strictly locally testable *an sich* but only when we take into account the admissible language where the annotation added to each surface token is bounded.

[3] A revised version of this paper has been accepted for publication in *International Journal of Foundations of Computer Science* (Yli-Jyrä, 2005b). It contains a slightly wider and more structured discussion on these classes of graphs.

[4] A comparison to different situations can be made. In the general case where the number of automata and their size bound are not constants, emptiness of intersection of automata is in *polynomial space* (PSPACE) (Kozen, 1977), and in *nondeterministic polynomial time* (NP) if one of the automata is acyclic. When the number n of automata is fixed, the problem becomes solvable in polynomial time according to the largest size of automata. If some of the automata are fixed, their sizes are constant, and they affect only the coefficient of the polynomial. If all but one automaton are fixed, the state complexity becomes a linear function.

homomorphism with the grammar retains the linear time complexity, because the resulting parser will be a finite-state transducer.

Estimating the State Complexity of an Example Grammar As it is now (at the latest) obvious that the time and the state complexities of FSIG are linear to the size of the sentence automaton, I turn my attention to the size of the coefficient. Obviously, any large-scale FSIG would correspond to a huge automaton. It has been estimated that such FSIGs as Eng-FSIG may need even 10^{1000} states when represented as a minimal deterministic automaton (Tapanainen, 1997). This is a very rough estimate, motivated just on the basis of the fact that a grammar contain some thousand more or less diagonal rules, each of which compiles into an automaton having typically 2 - 5 states or sometimes even thousands of states. The experiments carried out by Tapanainen (1992, 1997) suggest that combining all Eng-FSIG constraint automata into a single (non-symbolically represented) DFA would be impossible in practice.

Estimating the State Complexity of A Class of Grammars It is also important to understand how the complexity of the grammar is related to the way the grammar is designed. This area is perhaps most interesting because it may give some insight into the state complexity of various grammatical constraints and represented dimensions such as embedding, tree rank, the number of rules and categories, *etc*.

State complexity of set intersection (by means of minimal deterministic automata) has been studied in (Ravikumar, 1990; Leiss, 1991; Birget, 1991b,a, 1992; Yu and Zhuang, 1991). Currently there is a large body of literature about state complexity of basic finite automata operations. However, extended regular expressions used in NLP applications have not been addressed by automata theorists in these studies well enough. As far as regular expressions in FSIG are concerned, I have published some initial results on bracketing restriction (Yli-Jyrä, 2003d, submitted 2003) and context restriction operations (Yli-Jyrä and Koskenniemi, 2004), but there is certainly a need for further studies. According to our experiments (Yli-Jyrä and Koskenniemi, 2004), the size of each automaton obtained from context-restriction rules grows, in the worst case, exponentially according to the maximum depth of nested bracketing, and probably also to the number two-sided contexts.

The Lack of Insight into Locality and Parallelism in FSIGs Applying the state complexity analysis to FSIGs gives some information on the relative size of the coefficient hidden in their deterministic time complexity. However, I need substantially different kinds of insights in order to make the design of efficient FSIG parsers possible. I argue in the following that the insight we have lacked most is the understanding of the hidden structural locality in FSIGs.

For purposes of parsing where we often need to build the compact representations for ambiguity, it is extremely important to understand locality in the grammars that we parse. For example, the state of the art context-free parsers make use of tree locality in the derivation trees in order to present a Catalan number of trees (Church and Patil, 1982) in a polynomial ($O(n^2)$) space according to the length of the parsed surface string. The tabular representations for ambiguity that are involved in the parsers are

obviously related to Gaifman graphs describing the first-order properties of structures (Immerman, 1999) and the quantifier rank, but this connection has not been extensively studied in computational linguistics.

Another obvious and important application of locality comes up when we parallelize and solve problems using multi-processor computers. Problems that parallelize very well have locality properties that make this possible. So, in order to study locality in FSIG, we could try to study its parallel computational complexity.

In other words, there is a tremendous need to study locality and parallel computational complexity of FSIGs. Although these questions are more abstract than state complexity issues, the answers do have a close relationship to coefficients of deterministic time complexity as well. These connections can inspire new ways of looking at parsing in the FSIG framework.

Studying Parallelism by Means of Descriptive Complexity How the parallel computational complexity of FSIG parsing could be analyzed without assumptions on computational models?

The computational complexity can be understood as the richness of a formalism needed to specify the problem. The close relationship between computational complexity of problems and the richness of languages needed to describe them was established when Ron Fagin (1974) showed that the complexity class NP (the problems computable in nondeterministic polynomial time) is exactly the class of problems describable in existential second order logic. Today, this approach to computational complexity is called *descriptive complexity*[5]. Neil Immerman (Immerman, 1999, p.2) summarizes the role of descriptive complexity as follows:

> *It [descriptive complexity] gives a mathematical structure with which to view and set to work on what had previously been engineering questions.*

Illustration

Descriptive complexity has already characterized many complexity classes that are shown in figure 3.1. Immerman (1999) gives precise definitions for all these complexity classes, but I include here, for the reader's convenience, rough definitions for the classes of NP and co-NP. FO is the set of first-order boolean queries. It corresponds exactly to LH *i.e.* the class of boolean queries computable in $O(\log n)$ time by alternating Turing machines with restriction to a bounded number of alternations between existential and universal states. While FO uses a bounded number of simple variables, classes FO[$(\log n)^{O(1)}$] and FO[$n^{O(1)}$] use, respectively, poly-logarithmic and polynomial number of extended variables that are queried using predicate BIT$(i : j)$. BIT$(i : j)$ holds if and only if bit j in the binary representation of i is 1. FO(TC) is the closure of FO queries with arbitrary occurrences of reflexive, transitive closures of a binary relation, and FO(DTC) is its restriction to deterministic transitive closures. FO(DTC) and FO(TC) correspond, respectively, to LOGSPACE and NLOGSPACE.

[5] In different contexts, descriptive or descriptional complexity of formal systems may also mean *e.g.* state complexity of regular languages.

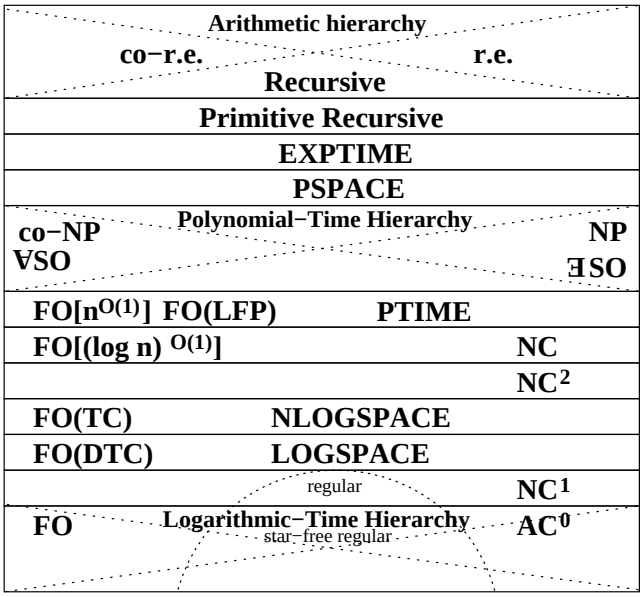


Figure 3.1: The world of descriptive complexity (*c.f.* Immerman, 1999).

The class LOGSPACE is the set of boolean queries computable in logarithmic space by a deterministic, multi-tape Turing machine. Class NLOGSPACE is the sets of boolean queries computable in logarithmic space by a nondeterministic, multi-tape Turing machine. Class PTIME is the sets of boolean queries computable in polynomial time by a deterministic, multi-tape Turing machine. PTIME corresponds exactly to class FO(LFP), the closure of first-order logic under the power to make inductive definitions. NC^k is the set of boolean queries accepted by a uniform sequence of bounded-depth k, *binary fan-in* circuits whose size is polynomial according to the size of the input, and AC^k if the set of boolean queries accepted by a uniform sequence of bounded-depth k, *unbounded fan-in* circuits whose size is polynomial according to the size of the input. AC^k is included to NC^{k+1}, and NC is the union of all NC^k where $k \geq 0$. The definitions for the non-PTIME complexity classes are not quoted here.

One of the strengths of descriptive complexity is its ability to analyze locality and structure of parallel time complexity classes. When we are restricted to regular languages, we find connections from logic (Thomas, 1997) and descriptive complexity classes to many varieties of star-free regular languages that have been studied in algebra (Pin, 1986) and in category theory (Eilenberg, 1974). This world will be opened to us when we turn our attention to the question of star-freeness in FSIG and show that this property holds for many interesting types of FSIG.

After this motivation, we are ready to look into the contributions more carefully with slightly more verbose comments given to each article.

3.2 Star-Freeness of Eng-FSIG Constraints

Article

"Describing syntax with star-free regular expressions," in *EACL 2003*.

3.2.1 Plan

This article, like the two other articles, are treated here with the following structure. First, the Introduction contains the *Summary*, *Motivations* and *Definitions*. It is followed by sections of *Main Results* and *Relevance*, and finally a section for *Other Useful Results*. The results sections often include *Remark*s that connect the result to the literature or clarifies it. Some results are accompanied with an optional *Illustration* section.

3.2.2 Introduction

Summary

This paper proves star-freeness of Eng-FSIG. On the way to this result, regular expressions are investigated and new methods for representing and processing grammar constraints are discovered.

Motivations

According to the abstract of the paper, the star-freeness result presented in this article is an essential *improvement* to the *descriptive complexity* of English Finite State Intersection Grammar.

Definitions

What does star-freeness mean? Extended regular expressions[6] use concatenation, empty set, complement, union and concatenation closure. The star-height of extended regular expressions is zero if it does not contain concatenation closures (the Kleene star or plus). The extended (or generalized) star-height of a regular language is zero, if it can be expressed using an extended regular expression whose (extended *i.e.* generalized) star-height is zero, *i.e.* with star-free extended regular expressions. Star-free (regular) languages are the regular languages whose extended[7] star-height is zero.

[6] There are many kinds of extended regular expressions (for example the FSIG regular expressions that include the context restriction operation) but this is the oldest and the most standard meaning. Sometimes extended regular expressions are termed generalized regular expressions.

[7] The attribute *extended* or *generalize* is important in order to make the difference to the restricted star-height (Eggan, 1963). The adjective *star-free* here as well as in my articles is equivalent to adjectival phrases *extended star-free* or *generalized star-free* whenever I do not explicate whether I mean the restricted star-height or not.

3.2.3 Main Results

(I) Expressibility in FO$[<]$

Theorem 1. Eng-FSIG as a whole is a regular expression that describes a star-free regular language.

Note that this result concerns the one-level definition of Eng-FSIG. It implies that the constraints in Eng-FSIG is definable in *first-order logic with linear order* (FO$[<]$). The first-order expressibility of FSIG is an essential improvement over the *monadic second-order logic* (MSOL) expressibility of FSIG.

A Remark It is well known that the satisfiability problem for first-order logic is in general undecidable, even for finite structures, but not for finite sets. This might worry some readers. However, the satisfiability problem for MSOL is decidable for strings, trees and other context-free graphs. MSOL for finite structures is called weak. Regular string sets are exactly those string sets definable with *weak monadic second-order logic with one successor* (wMSOL$[S]$) (Büchi, 1960; Elgot, 1961).

The FO$[<]$ definable languages are a subset of wMSOL$[S]$ definable languages in a similar way as star-free languages are a proper subset of regular languages. The result that Eng-FSIG defines star-free *i.e.* a FO$[<]$ definable language is, thus, a genuine improvement over the descriptive complexity of the general FSIG.

(II) New Restriction to the FSIG Framework

In the section Discussion of the article, I wrote as follows:

> *Finally, the main contribution of this paper is to show that ENGFSIG describes a star-free set of strings. It seems probable that this narrowing could be added to the FSIG framework in general.*

In the subsequent papers, I kept in mind that the constraints in a grammar that belongs to this new class should be easily reducible to star-free regular expressions. I think that the idea of star-freeness was a handy view that helped in analyzing what kind of relations various constraints should express.

A Remark It is interesting to note that Kornai (1985) has argued that natural languages are non-counting *i.e.* star-free regular languages. According to him, this restriction might be useful in language acquisition:

> *Hopefully, this special position of NLs in the Chomsky hierarchy can be utilized in streamlining the (oracle) algorithms modeling language acquisition, because such algorithms (if actually implemented) would greatly simplify the descriptive work of the linguist, and, at least to a certain extent, would finally fulfil the structuralists' promise of discovery procedure.*

(III) Grammar Building Blocks that Maintain Star-Freeness

The article identified some idioms in regular expressions that are useful when we write other star-free FSIGs:

- *limited balanced bracketing* (LBB)[8] expressed through a parameterized star-free language,
- context restriction operation,
- languages like X^* where X is an alphabet, and
- marked iterations of a language.

A Remark It is perhaps necessary to make clear that the article is not proposing star-free regular expressions as a tool for those who write linguistic descriptions. I am not against extending the linguist's grammar formalism with various syntactic extensions, and using the Kleene star. Syntactic extensions to regular expressions are desirable and even necessary because they can hide technical details and make the user interface more intuitive. In Introduction I wrote:

> *Regular expressions in FSIG can be viewed as a grammar-writing tool that should be as flexible as possible. This view point has led to [the] introduction of new features into the formalism. — A complementary view is to analyze the properties of the languages described by FSIG regular expressions.*

3.2.4 Relevance

In the sequel, I try to answer to the question "why star-freeness is important?".

Star-free FSIGs Define a Natural Class of Languages In addition to star-free extended regular expressions, the star-free regular languages can be defined using many other equivalent ways that characterize the same class of languages. The alternative characterizations for star-free languages include the following:

- counter- or permutation-free finite automata (McNaughton and Papert, 1971),
- finite aperiodic monoids (Schützenberger, 1965),
- cascade product of reset semi-automata (Meyer, 1969),
- languages definable with FO$[<]$ (McNaughton and Papert, 1971),
- linear temporal logic (*c.f.* Perrin and Pin, 2001),
- the smallest variety closed under the pure star operation (*c.f.* Pin, 1986),
- the concatenation hierarchies (*c.f.* Pin, 2003),

[8] The term LBB does not occur in the article.

- first-order string automata, (Schwentick and Barthelmann, 1999),
- loop-free alternating automata (Salomaa and Yu, 2000), and
- the forbidden-pattern hierarchy (Glaßer, 2001).

Star-free FSIGs Define Local Properties The expressibility in FO$[<]$ opens new scenarios for FSIG parsing, because first-order properties are local. For example, according to a famous theorem by W. Hanf (Immerman, 1999, p.102-103), first order sentences with a *bounded quantifier rank*[9] cannot distinguish between two graphs of bounded degree if the graphs have the same number of local neighborhoods of all possible types where the number of possible types depends exponentially on the quantifier rank [10]. It is, thus, no wonder that dependency on the neighborhoods under the successor or linear precedence relation is the basis for definition of many special classes of regular languages (McNaughton and Papert, 1971) and for their recognition with tabulating methods.

Star-free FSIGs Have Lower Computational Complexity According to Immerman (1999), the following complexity classes are equivalent:

AC^0 = FO = LH = CRAM[1]

I explained the meaning of AC^0, FO and LH already on page 25. The less known complexity class CRAM$[1]$ is the set of boolean queries computable in constant parallel time on a CRAM. A *concurrent random access machine* (CRAM) is a machine that consists of a large number of processors, all connected to a common, global memory. At each step, any number of processors may read or write any word of the global memory. If several processors try to write the same word at the same time, then the lowest numbered processor succeeds.

Among all regular languages, there are some that do not belong to AC^0, but all are included in NC^1 (NC^1 is the set of boolean queries accepted by a uniform sequence of *binary* polynomial-size circuits). AC^0, FO, LH and CRAM[1] are proper subclasses of NC^1. AC^0 *contains* all star-free regular languages (Thomas, 1997) and, therefore, we actually see that star-free languages have a smaller computational complexity than regular languages in general.

Star-free FSIGs Have Certain Compact Representations We can make use of parallelism even in conventional computers:

- We can represent the grammar in a fashion that makes use of parallelism. The article suggests that a compact grammar representation would be based on *loop-free alternating finite automata* (LF-AFA) by (Salomaa and Yu, 2000).

[9] The quantifier rank of a first-order formula is basically the number of nested quantifiers in it.

[10] This result is perhaps easier to understand if we recall that there are only finitely many inequivalent formulas of quantifier rank r (Immerman, 1999, p.96).

- A very good compact representation would also allow the generation of a parse forest in a manner that resembles parallelism, by turning parallelism of recognition into dynamic programming in the parse forest generation. In chapter 5 we will, in fact, suggest a representation where the parsing would be carried out in linear time with a constant number of parallel processed subgrammar layers. Each of these layers would represent some of the local string neighborhoods in a very compact manner.

So far, we still understand only a surface of the relationship between FO expressivity and compact representation of ambiguity in parsing. It might turn out that circuit depth or parallel time — although they are constants for each language in CRAM[1] and AC^0 — corresponds to the depth of a tree that describes how sub-grammars should be joined or made consistent during parsing.

3.2.5 Other Useful Results

The Star-Freeness of LBB

The LBB language recognized by the automaton of figure 3.2 is star-free. The same language can be described using the following definition (the definition is not star-free) for the corresponding star-free regular language $\boxed{\cdots}^d \subseteq \Sigma^*$, where $\texttt{@<}, \texttt{@>}, \texttt{@} \in \Sigma$:

$$\boxed{\cdots}^d = \begin{cases} \Sigma^* - \Sigma^*\{\texttt{@<}, \texttt{@>}, \texttt{@@}\}\Sigma^* & \text{if } d = 0 \\ (\boxed{\cdots}^{d-1} \ \cup \ \texttt{@<}\boxed{\cdots}^{d-1}\texttt{@>})^* & \text{when} d > 0. \end{cases}$$

The article itself contains a star-free regular expression for this language. The star-freeness of $\boxed{\cdots}^d$ was not at all obvious to me. I used the AMORE program to prove the star-freeness of $\boxed{\cdots}^d$. The automatically obtained star-free expression was huge. For illustrative purposes, I needed a shorter one which I found through a trial-and-error approach. Similar languages had been studied in connection to the DDH of star-free languages, but I was not aware of this connection at the time of publication of this article.

Remarks on Context Restriction

It turned out later when I was writing the article that I identified a difference in the XFST implementation of the restriction operation and my new formula that did not rely on transducers or substitutions. These results led to a further publication (Yli-Jyrä and Koskenniemi, 2004) where the different implementations are documented as

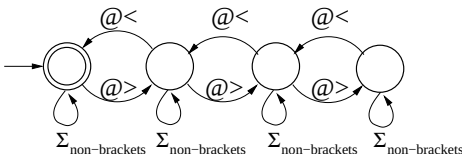


Figure 3.2: An automaton for an LBB like in Eng-FSIG.

carefully as well as possible. When testing the differences, the regular expressions of Voutilainen's Eng-FSIG constituted a really useful material.

3.3 Approximations of Context-Free Grammars

Article

> "Regular approximations through labeled bracketing (Yli-Jyrä, submitted 2003)"

This article has been submitted to the post-proceedings that should become an online proceedings published by CSLI Publications. A precursor of this heavily revised article was presented in FGVienna 2003 (Yli-Jyrä, 2003d).

3.3.1 Introduction

Summary

The article introduces FCFBGs and their approximations that constitute a new type of FSIGs.

Motivations

In general, implementing good parsers of star-free FSIGs, such as Eng-FSIG, is a difficult task because very little is known about the class of constraints that can be used in the grammar — star-freeness is still a very rough characterization. However, if we consider some well-understood and restricted kinds of FSIGs, we may have better chances of understanding the structure of the grammar and of finding an efficient parsing algorithm. As a promising alternative, we could get FSIGs mechanically from context-free grammars by approximating them.

Definitions

When we say that we approximate context-free grammars we mean that it is the SGP of CFGs that is approximated. Some other approximations are more suited for approximation of the languages generated by context-free grammars (WGP of CFGs) than for obtaining an approximation for sets of parse trees (SGP of CFGs).

For us, constituent X is *center embedded* to a larger constituent Y if the yield of Y is of the form $\alpha X \beta$, where α and β are nonempty. Such a center-embedding is a *self-embedding* if both X and Y are of the same category[11]. If center-embedded constituents are nested, all of them do not have to be of the same category (*i.e.* the self-embedding depth may be lower).

[11]The terms self-embedding and center-embedding are often used interchangeably. If we talk, for example, about center-embedding of S we actually talk about self-embedding as well, because S is assumed to be also the root.

Extended context-free grammars (ECFG) are an extension of CFGs. In ECFGs, right-hand sides of normal CFG productions are replaced with regular expressions. Due to this extension, the rules of ECFG that are called production schemas.

Dyck languages in the article are actually also semi-Dyck languages. Because the class of Dyck languages contains the semi-Dyck languages as a subset, semi-Dyck languages are often imprecisely referred to as Dyck languages. In Dyck languages, the only requirement is that the number of opening brackets is equal to the number of closing brackets, while in semi-Dyck languages each opening bracket must also match a closing bracket that is on the right (Harrison, 1978).

3.3.2 Main Results

(I) An Approach for Obtaining FSIG Approximations

In this article, FSIG is presented for the first time as a framework for representing regular approximations of non-regular grammars. The approach consists of

- encoding structures of the original grammar through brackets,
- representation of the surface string language as a homomorphic image of the set of the bracketed deep strings,
- describing the set of all bracketed strings by a derivative of a semi-Dyck language,
- representation of the grammar through a "Chomsky-Schützenberger style representation" where the semi-Dyck derivative is combined with finite languages using concatenation and boolean operations,
- Wrathall's technique for obtaining any semi-Dyck language D_k from the semi-Dyck language D_1, and
- star-free approximation of the semi-Dyck language D_1.

This same approach has been used later in two other approximations that are included in this dissertation.

A Remark The original Chomsky-Schützenberger representation[12] is not linguistically appealing as such. Also, it is not based on a fine-grained set of assertions such as in MTS (it compiles the original productions into a single right-linear grammar that describes the string local properties of the so-called *standard context-free language* that is obtained from the grammar). Furthermore, it is intended for representing surface languages only. If we also want to obtain phrase-markers — bracketed strings representing derivation trees — we need to do some extra work.

[12]The set of context-free phrase-structure trees can be captured with a bracket string language – a so-called semi-Dyck language. Chomsky and Schützenberger (1963) presented a theorem according to which each context-free language can be represented as a homomorphic image of an intersection of a semi-Dyck language and a regular language.

A virtue of our approach is that it uses the string encoded descriptions of linguistic structures directly which means that we do not need to simplify the bracketing used in the grammar representation in order to obtain the desired structural descriptions. Moreover, our representation can be adapted for reduced bracketing.

(II) Reducing Tree-Locality into Star-Free Constraints

The article shows that when the height of derivation trees is limited, the local trees correspond to regular set properties of bracketed strings. Under the assumption that the right-hand sides of the approximated *context-free bracketing grammar*s (CFBGs) represent star-free regular languages, the properties limited of bracketed strings encoding local trees are not only regular, they will actually correspond to star-free sets.

An Illustration Star-free approximations can specify balanced bracketed strings from inside (bottom-up) even if the the total depth of bracketing is not known. However, the same does not hold for from outside to inside (top-down). Rather, we need to know the limit for nested brackets (of the same type). This is illustrated as follows. According to the article, the bracketing restriction constraint $\#Lc\_\_Rc\# \Rightarrow f_2(E)$ (page 10), can be approximated through the formula (on page 11)

$$V_T^* \; - \; Lc \; (A_k - f_3(f_2(E))) \; Rc.$$

According to this formula, a substring between an *Lc*-prefix and an *Rc*-suffix should represent a well formed subtree. However, it is not possible to use the following formula to restrict the top side of the tree because the *Lc* prefixes and the *Rc* suffixes match each other in order to constitute well-formed contexts for subtrees.

$$A_k \; - \; Lc \; (V_T^* - f_3(f_2(E))) \; Rc.$$

The first formula is a bottom-up alternative. The second formula, that was intended to be the top-down alternative, fails to define a tree property, because V_T^* is not a set of trees.

A Remark Peters and Ritchie (1969) showed that the derivation trees of CFGs can be described through context sensitive rules that acted as constraints[13] rather than productions. My bracketing restriction constraints extend this old idea to unranked context-free grammars. I have realized that the pre-proceedings version of my article (Yli-Jyrä, 2003d) involved some fundamental problems because the result of Peters and Ritchie was limited to ranked context-free trees.

It would be interesting to see in future, whether we could approximate also some non-tree-local tree properties (*e.g.* proper analysis predicates (Joshi and Levy, 1982), and logical constraints (Rogers, 1998)) with star-free constraints on bracketing.

[13]These constraints *analyze* or validate the context-free trees. I used the historical term *to analyze* in the title of (Yli-Jyrä, 2004a) in order to indicate the connection to this approach.

(III) The Connection to the Dot-Depth Hierarchy

The article points out the connection to the DDH (Brzozowski and Knast, 1978) of regular approximations. Although the class of star-free languages corresponds exactly to this hierarchy I may be the first to carry DDH over to computational linguistics and to regular approximations in particular. I believe that the dot-depth of grammars measures their parsing complexity in an interesting way.

The DDH of star-free subsets of Σ^* can be defined inductively as follows. The first level, $\mathbf{B}_0$, contains subsets of $\Sigma \cup \{\epsilon\}$. Every further level $\mathbf{B}_i$, $i > 0$, is obtained from the languages of $\mathbf{B}_{i-1}$ by closing them first under concatenation, and then closing the resulting family of languages under union and complementation.

The dot-depth measure is not generally available for regular languages. When I made the restriction to star-free FSIGs, I actually made the measure of dot-depth available. Unfortunately, determining the exact dot-depth of a star-free regular language is, in general, a difficult problem (Pin, 2003).

A Remark The DDH has to do with complementation in star-free expressions and quantifier alternations in the corresponding first-order description. Thomas (1982) showed that there is a very natural way in which the DDH corresponds to the quantifier alternation hierarchy. This classical hierarchy of first-order logic is based on alternation of existential and universal quantifiers: a star-free language is of dot-depth d if and only if it is defined by a boolean combination of L-formulas (L is a first order logic in which star-free languages are described) in a *prenex normal form* (PNF)[14] with a Σ_d prefix.

Star-free FSIGs make heavy use of complementation. For example in Yli-Jyrä and Koskenniemi (2004), I define context restrictions and bracketing restrictions using generalized restrictions. On one hand, the generalized restriction is defined using a regular expression with a double complementation, and on the other hand, context restrictions (*ibid.*) correspond to a first-order formula, where existential and universal quantifiers alternate. Furthermore, alternations occur also in the generalized definition of Eng-FSIG constants that express LBB languages (Yli-Jyrä, 2003a). In the current article (Yli-Jyrä, submitted 2003), I realised that automata similar to the one shown in figure 3.2 are similar to those that are used to establish the infinitude of the DDH.

3.3.3 Relevance

State Complexity and Dot-Depth The usual way to measure the complexity of regular languages is their deterministic state complexity, but I started by studying the star-freeness property. The dot-depth of star-free languages also gives a useful view on state complexity.

It is a well known result of Meyer (1975) that when a wMSOL$[S]$ formula is compiled into an automaton, the worst-case state complexity of the result is non-elementary according to the length of the formula (Klarlund, 1998). What does this mean? A *non-elementary* function grows faster than exponentially: when its argument grows, the

[14] A formula is in prenex normal form if it consists of a string of quantifiers applied to a quantifier free formula.

next value of the function will infinitely often be exponentially larger than its previous value.

The result that is available for wMSOL$[S]$ could be compared to what happens when we compile first-order formulas into deterministic automata. However, the author is not aware of results that would associate non-elementary state complexity with FO$[<]$, although there is some related results (*c.f.* Frick and Grohe, 2004). We note also that each level of the DDH concatenates at least two languages of a lower level. Concatenation of DFAs of two languages has, in general, exponential state complexity (Yu, 1999) and exponential blow-up is possible even when star-free languages are concatenated (Mohri, 1995). We can expect, thus, that dot-depth hierarchy is related to rapidly growing – if not non-elementary – state complexity. Perhaps further studies (or missing references) on state-complexity would be in place.

In degenerate cases, such as A_k (figure 1 in the article), the state complexity is not non-elementary nor exponential but linear to the dot-depth. So, in grammar representations using a semi-Dyck approximation we get first k levels in the DDH almost for free. When A_k is then used in the bracketing restriction rules we really have an exponential blow-up. Due to this, the state complexity of regular approximations discussed in this article is no worse than exponential to their dot-depth.

Another degenerate case would use different bracket labels for embedded phrases. This does not decrease the the state complexity, but makes an alternative, more succinct finite-state representations available as we will see in chapter 5.

3.3.4 Other Useful Results

(I) Reduced Bracketing Solves the Tail Recursion Problem

If the nesting of brackets is limited — as it is necessary for any regular approximation — full bracketing fails to capture unbounded left- or right embedding (it may still define infinite languages because the bracketing can represent unranked trees, but sometimes we do not want to reduce tail-recursion into iteration in the syntactic analysis). The article solves this incapability of bracketing grammars by introducing a reduced bracketing scheme to context-free bracketing grammars. This way to solve the encoding of tail recursion does not have anything to do with regular approximation although regular approximation will take advantage of the result, the reduced bracketing scheme.

A Remark Reduced bracketing is not a new thing (Krauwer and des Tombe, 1980, 1981; Johnson, 1996, 1998). According to Ejerhed and Church (1983), a similar representation was proposed by Ronald Kaplan (Church and Kaplan, 1981) in Xerox. In INTERLISP (Teitelman, 1978), super-parenthesis (a bracket ']') was used to close any number of open parentheses. Chomsky (1963, section 4) also suggests optimising brackets in certain contexts.

An Illustration Here is an example of reduced bracketing:

$$\begin{array}{ll} [[[[[a]b]c]d][e][[f]] & g[h[i[[j[k[l[m]]]]n]]]] \\ [\quad a\rangle b\rangle c\rangle d\rangle[e][\langle f\ ] & g\langle h\langle i\langle[j\langle k\langle l\langle m\quad]n\quad] \end{array}$$

The example shows two bracketed strings: a fully bracketed one and a reduced one. They can be transformed into each other in linear time using a deterministic two-way stack machine.

(II) Definitions for Bracketing Grammars

The paper introduces two kinds of non-regular grammars for bracketed strings: CFBGs and FCFBGs.

CFBGs are a subclass of so-called ECFGs. They have a lot in common with bracketed context-free grammars (Ginsburg and Harrison, 1967), but also some important differences that makes this new definition necessary. CFBGs generate context-free sets of bracketed string where the bracketing in strings represents (possibly) unranked derivation trees. CFBGs have been defined in such a way that it is easy to make some modifications to the bracketing scheme and, especially, to use reduced bracketing in some productions.

The main difference between FCFBGs and CFBGs is that FCFBGs do not use a rewriting mechanism. Thus, FCFBGs can be seen as an MTS grammar, while CFBGs is a GES grammar.

A Remark The following discussion argues that the proposed grammars are relevant to linguistics and computer science. Ejerhed and Church (1983) advocate the use of extended context-free grammars for flat syntax. Although CFGs have been largely replaced by mildly context-sensitive grammars as the syntactic theory in the state of the art of NLP, CFGs have still a lot of practical relevance and, for example, probabilistic CFGs have been used recently in treebank construction. Context-free trees capture, for example, the topological structure or recursive chunks in German syntax, as well as models information structures where *e.g.* Theme, Rheme, Background and Focus in the sentence form a relatively shallow recursive structures. Eng-FSIG is related to CFBG because the brackets (@<,@>) in Eng-FSIG can mark flat clause structures whose tree-rank is unbounded. Furthermore, combining a FCFBG with a dependency-based FSIG in a multi-tape finite-state system would provide a way to implement a topological dependency grammar.

Bracketing restrictions have deepened our understanding of the possible restrictions and inspired our study of their state complexity. Moreover, some computer science applications for these grammars may be found in XML (and SGML) processing, where the parse trees may be unranked and the markup may miss some tags (like in SGML document types). FCFBGs might be extended, for example, in a way that corresponds to regular frontier check in tree automata (Jurvanen et al., 1993). In fact, Joshi and Levy (1982) advocate the use of local tree constraints in linguistics, some of which (proper analysis predicates) might now be represented with bracketing restrictions in FCFBGs. This might imply that FCFBGs can define sets that cannot be defined with CFBGs.

Finding this connection to CFGs was very instructive to the author. It helped later to relate FSIG parsing with standard parsing techniques. It also led to the question of encoding of projective dependency grammars. Moreover, the implications of Chomsky-Schützenberger style representations turned out to be deeper than expected, extending

	full bracketing	reduced bracketing
BCFG	$S \rightarrow [_S\ NP\ VP\ ]_S$ $VP \rightarrow [_{VP}\ V\ ]_{VP} \mid [_{VP}\ V\ NP\ ]_{VP}$ $NP \rightarrow [_{NP}$ Jim $]_{NP} \mid [_{NP}$ Sue $]_{NP}$ $V \rightarrow [_V$ ran $]_V$	$S \rightarrow [_S\ \overleftarrow{NP}\ \overrightarrow{VP}\ ]_S$ $\overrightarrow{VP} \rightarrow \langle_{VP}\ \overrightarrow{V} \mid \langle_{VP}\ V\ \overrightarrow{NP}$ $\overleftarrow{NP} \rightarrow$ Jim $\rangle_{NP} \mid$ Sue $\rangle_{NP}$ $\overrightarrow{NP} \rightarrow \langle_{NP}$ Jim $\mid \langle_{NP}$ Sue $\overrightarrow{V} \rightarrow \langle_V$ ran $V \rightarrow [_V$ ran $]_V$
Flat BCFG	$\#\_\_\#$ ⇒ $[_S\Delta]_S$ $[_S\ \_\_\ ]_S$ ⇒ $[_{NP}\Delta]_{NP}[_{VP}\Delta]_{VP}$ $[_{VP}\ \_\_\ ]_{VP}$ ⇒ $[_V\Delta]_V \mid [_V\Delta]_V[_{NP}\Delta]_{NP}$ $[_{NP}\ \_\_\ ]_{NP}$ ⇒ Jim | Sue $[_V\ \_\_\ ]_V$ ⇒ ran	$\#\_\_\#$ ⇒ $[_S\Delta]_S$ $[_S\ \_\_\ ]_S$ ⇒ $\Delta\rangle_{NP}\langle_{VP}\Delta$ $\langle_{VP}\ \_\_\ B_R$ ⇒ $\langle_V\Delta \mid [_V\Delta]_V\langle_{NP}\Delta$ $B_L\ \_\_\rangle_{NP}$ ⇒ Jim | Sue $\langle_{NP}\ \_\_\ B_R$ ⇒ Jim | Sue $\langle_V\ \_\_\ B_R$ ⇒ ran $[_V\ \_\_\ ]_V$ ⇒ runs|hits
Context restrictions	$(\{$Jim,Sue,runs,hits$\} \cup B_L \cup B_R)^*$ Jim | Sue ⇒ $[_{NP}\ \_\_\ ]_{NP}$ $[_{NP}\Delta]_{NP}$ ⇒ $[_{VP}[_V\Delta]_V\ \_\_\ ]_{VP} \mid [_S\ \_\_\ [_{VP}\Delta]_{VP}]_S$ runs|hits ⇒ $[_V\ \_\_\ ]_V$ $[_V\Delta]_V$ ⇒ $[_{VP}\ \_\_\ ]_{VP} \mid [_{VP}\ \_\_\ [_{NP}\Delta]_{NP}]_{VP}$ $[_{VP}\Delta]_{VP}$ ⇒ $[_S[_{NP}\Delta]_{NP}\ \_\_\ ]_S$ $[_S\Delta]_S$ ⇒ $\#\_\_\#$	$(\{...,$hits,$\langle_{NP}, \rangle_{NP}, \langle_{VP}, \rangle_V\} \cup B_L \cup B_R)^*$ Jim|Sue ⇒ $\langle_{NP}\ \_\_\ B_R \mid B_L\ \_\_\rangle_{NP}$ $\rangle_{NP}$ ⇒ $[_S\Delta\ \_\_\ \langle_{VP}\Delta]_S$ $\langle_{NP}$ ⇒ $\langle_{VP}[_V\Delta]_V\ \_\_\ \Delta B_R$ runs|hits ⇒ $[_V\ \_\_\ ]_V \mid \langle_V\ \_\_\ B_R$ $\langle_V$ ⇒ $\langle_{VP}\ \_\_\ \Delta B_R$ $[_V\Delta]_V$ ⇒ $\langle_{VP}\ \_\_\ \langle_{NP}\Delta B_R$ $\langle_{VP}$ ⇒ $[_S\Delta\rangle_{NP}\ \_\_\ \Delta]_S$ $[_S\Delta]_S$ ⇒ $\#\_\_\#$

Table 3.1: Comparison of different ways to represent the same tree set

to some *mildly context-sensitive grammar*s (MCSGs) in contributions discussed in the next chapter. Learning about bracketing schemes for context-free constituents was useful for understanding more complex representations.

An Illustration Table 3.1 illustrates two kinds of bracketing languages and three kinds of bracketing grammars. Both languages — a full bracketing language and a reduced bracketing language — represent the same set of trees through different kinds of bracketed strings. These two languages are described with three different grammars: a CFBG, a FCFBG and a grammar that uses context restriction. The last one is included here for the sake of completeness as an *illustration* that shows that bracketing restriction constraints are not the only way to validate bracketing. However, it is interesting to see that only grammars with bracketing restrictions allow for rules that resemble CFGs rules. It should be noted that FSIG approximations are obtained from these last four grammars by replacing Δ with a regular language.

3.4 Approximations of Dependency Grammars

Article

> "Approximating dependency grammars through intersection of regular languages" (Yli-Jyrä, 2005a)

A precursor of this revised article was presented in the pre-proceedings of CIAA 2004 (Yli-Jyrä, 2005a). A further extension has been accepted for publication in *International Journal of Foundations of Computer Science* (World Scientific) in a CIAA follow-up issue. The journal version emphasizes star-freeness of the approximations and discusses the structural properties of dependency trees more carefully.

3.4.1 Introduction

Summary

The article introduces an approach for full (projective) dependency syntax within the FSIG framework.

Motivations

I am not aware of any finite-state approaches to dependency parsing that is based on intersection or assertions, although approaches with more powerful constraint programming techniques (*e.g.* Duchier, 1999; Maruyama, 1990) were available and FSIG approximations existed for context-free grammars (Yli-Jyrä, submitted 2003).

Definitions

A D-tree or a dependency graph (D-graph) is said to be *semi-planar* (or *planar* according to Sleator and Temperley, 1991) if the links do not cross each other when drawn above the sentence. The sentence line drawn on a plane splits the plane into two halves, thus we should have the *semi-* prefix. For example, the D-tree in figure 3.3 is semi-planar. The article specifies the projective graphs as a subset of the semi-planar D-graphs.

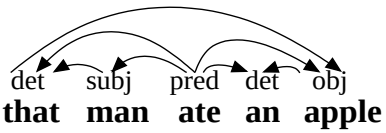


Figure 3.3: A semi-planar D-graph (an imaginary situation).

3.4.2 Main Results

(I) A New Class of FSIGs

The article introduces a new class of FSIG: approximations of dependency grammars. Star-freeness of these regular approximations can be proven easily using the heuristics developed earlier.

The new class illustrates how classes of structural descriptions can be axiomatized using FSIG constraints. The idea of definitive constraints, axioms, evolve from "administrative" constraints used earlier in FSIGs (*c.f.* Chanod and Tapanainen, 1996b).

A Remark Although star-freeness of the resulting approximations (bracketed languages) is not mentioned in the paper, it is "inherited" along with the FSIG approach. Kleene stars are used in the regular expressions of the article in such a way that it is easy to reduce these regular expressions into a star-free format.

(II) String Locality of Rules

The article formulates dependency rules of HGDG as regular constraints. These constraints validate finite substrings that occur between two word boundaries (#) in the annotated strings[15].

A Remark String locality gives rise to a new parsing approach that was suggested in the paper. According to it, automata representing local lexical constraints[16] will be applied during parsing before automata that check more global properties such as balanced bracketing.

Furthermore, the new grammar representation makes it particularly easy to analyze the upper-bound for the dot-depth of the resulting grammar instances: the maximum dot-depth in the set of constraints is an upper-bound for the dot-depth of the whole set, because each level of the DDH is closed under intersection. Accordingly, we need only to consider the dot-depth of constraint (3b) and of those in (5), which have a very simple definition that depends on d, the limit for nested brackets. This points to the same direction as suggested in the case of FSIG approximations of FCFBGs: the state complexity would not be non-elementary although the dot-depth does not seem to be bounded.

The actual dot-depth can be lower than the upper bound for two reasons: (i) the described language can be empty or (ii) it does not necessarily happen at all that a dependency link embraces another link of the same category.

(III) A Novel Dependency Bracketing Scheme

The article presents a novel string-based representation for dependency trees.

[15] There are some constraints and constraint schemata that contain Kleene's closures, and are not, strictly speaking, string local. In these constraints, the closures could be replaced, however, with finite languages, as the constraints restrict substrings between word boundaries and these substrings are otherwise bounded, which is a combined effect of several constraints.

[16] Constraints whose domain of locality is bounded by two word boundaries (#).

A Remark This representation is closely related to a recent, but more complicated representation by Oflazer (2003), but in a strict contrast to traditional representations (Lecerf, 1961; Hays, 1964; Nasr and Rambow, 2004).

Oflazer (2003) has presented a representation that is closely related to ours. In contrast to it, the most previous string-based encoding schemes for dependency trees are based on phrase markers of context-free grammars. They would represent the parse tree as follows:

- According to Lecerf (1961):
  [([([that]) .[man]]) . [[ate].([([an]) .[apple]])]]
- According to Hays (1964):
  ⋆(pred(subj(det(⋆),⋆), ⋆, obj(det(⋆),⋆)))
- According to Nasr and Rambow (2004):
  $\lfloor_{\mathbf{pred}} \lfloor_{\mathbf{subj}} \lfloor_{\mathbf{det}}$ **that** $\rfloor_{\mathbf{det}}$ **man** $\rfloor_{\mathbf{subj}}$ **ate** $\lfloor_{\mathbf{obj}} \lfloor_{\mathbf{det}}$ **an** $\rfloor_{\mathbf{det}}$ **apple** $\rfloor_{\mathbf{obj}} \rfloor_{\mathbf{pred}}$

3.4.3 Relevance

The Significance of the New Bracketing Scheme

My new encoding scheme differs radically from the classical bracket-based notations for dependency trees. Choosing this new encoding is motivated by the following two arguments:

First, in the classical notation (Lecerf, 1961; Hays, 1964; Nasr and Rambow, 2004), the brackets for dependent nodes do not generally appear in the local neighborhood of their governors. In my encoding, labeled brackets for all the dependents of a governor word w are indicated close to w. Due to this locality, the HGDG rules over the bracketed representation can be expressed using regular languages.

Second, the representation is "almost" capable of encoding some non-projective trees and semi-planar graphs. Namely, the encoding can be extended to non-projective dependency trees and graphs by relaxing certain constraints. For example if I do not include the irreflexivity constraint, the outside-to-inside constraint and all Robinson's axioms, I get a much larger set of graphs.

Furthermore, this new bracketing scheme can be extended to non-projective D-trees and D-graphs. These possibilities are made explicit in (Yli-Jyrä, 2003c, 2004b,a; Yli-Jyrä and Nykänen, 2004). Non-projective D-graphs can be represented with strings by splitting them to semi-planar subgraphs each of which is then described by bracketed strings that share the same sequence of word nodes. These bracketed strings for semi-planar graphs can be written on different input tapes or they can be layered (or interlaced) to a single string, provided that each semi-planar graph is encoded with a different set of brackets.

3.4.4 Other Results

Axiomatization of Planar D-graphs and D-trees

The paper axiomatizes the following classes of graphs or their string representations:

- semi-planar graphs (axioms 1-5),
- acyclic semi-planar graphs (axioms 1-5,10,12,13), and
- projective dependency trees (all axioms).

Moreover, the paper comments on Jane Robinson's axiomatization and shows that her four axioms or the usual projectivity condition, as they stand, do not themselves imply acyclicity of the D-graph. In fact, acyclicity is not first-order expressible (Immerman, 1999, p.101), thus, not a local property, but, of course, some subsets of the acyclic graphs may be expressible in *first order boolean queries* (FO) logic.

In my representation, the context-free language for balanced bracketing enforces a necessary global condition (semi-planarity) for acyclicity. An important, but simple observation that is exploited in the paper is that even limited bracketing will enforce this condition to the represented graphs. When the number of nested links is limited, this condition becomes first-order expressible.

A Remark Projectivity condition has been studied perhaps first by Lecerf (Lecerf, 1960; Ihm and Lecerf, 1963)[17]. Differences between previously defined projectivity conditions have been studied by (Marcus, 1965, 1967).

Semi-planar graphs can be seen as an extension of trees. They are closely connected to proof-nets, although there are also proof nets that are not planar (Gaubert, 2004).

A Representation for Context-Free Languages

The article presents a flat representation for HGDG's. This can be used as a new Chomsky-Schützenberger style characterization for context-free languages. A peculiar property with my representation is that it starts from dependency grammars rather than from context-free phrase-structure grammars.

A Remark Correspondence between CFGs and dependency grammars and their integration into the same framework has been studied in (Robinson, 1967; Covington, 1994c,a; Höfler, 2002; Bohnet, 2003; Robinson, 1967; Teich, 1998; Nivre, 2003b; Dras et al., 2004).

An Open Problem Many deterministic dependency parsers (Arnola 1998; Elworthy, 2000; Nivre, 2003a; Nivre and Scholz, 2004) produce projective analyses. This is, however, not a necessity (Covington, 1990, 1992, 1994b, 2001). It is not clear how the output of these parsers could be characterized as a set of structures, because the deterministic parsers usually lack a declarative specification.

[17] According to Marcus (1965), the notion of projectivity appears in 1959 in the work of Harper and Hays, and the first projectivity-based dependency grammar was presented in May 1960 by D. E. Hays at the Rand Corporation, Santa Monica, California. H. Gaifman's 1965 paper had already appeared in 1961.

3.5 Conclusions

I have summarized articles that investigate three kinds of FSIGs: (i) Eng-FSIG, (ii) and approximations of FCFBGs and (iii) HGDGs. The articles contain, by themselves, new approaches and contributions to finite-state methods in natural language processing.

The Most Important Contribution

The class of star-free FSIGs is a new, important discovery that gives more insight into how some FSIGs are structured, how new FSIGs should be designed and what is the computational complexity of these grammars. The articles are tied to each other under the theme of star-freeness, although this was not made explicit in the selected version of the third article.

In this chapter, we have been asking what kinds of insights these articles can give about computational complexity of their grammatical model strings. The complexity of regular languages is, in any case, low, but descriptive complexity allows for defining finer distinctions for low complexity languages. When I showed, in the first article, that the model string set of Eng-FSIG is star-free, I actually obtained also a result on its computational complexity. The approach of regular expression equivalences as presented in the first article are also applicable to my new FSIGs, although I do not elaborate the details of their star-freeness proofs.

The star-freeness of sets of model strings for assertions implies that the model string languages are in DDH and LH. This means that the analysis of a finite-state approximation can be made more refined by studying its dot-depth, the number of quantifier alternations in its first-order description, or the number of nested complementation needed in the star-free regular expression. Finally, I argued for that dot-depth is connected to worst-case deterministic state complexity of star-free grammars.

Chapter 4

Linguistic Applicability

4.1 Orientation

4.1.1 Summary

This chapter will present an approach to modeling complex dependency-based structural descriptions using FSIGs.

4.1.2 Problem

The Quest for Appropriate Grammars Defining a new class of grammars that could give appropriate syntactic descriptions for natural language sentences is not an easy task. Different hypotheses for possibly adequate formal grammars exist, ranging from finite-state models to certain context-sensitive classes of grammars, and contextual grammars. Currently the best characterizations for natural language syntax is given in terms of mildly context-sensitive grammars and linear context-free rewriting systems. At the same time, these models are challenged by an evergreen approach: dependency syntactic grammars that are not limited to projective constituent structures but associate words according to their valencies.

How does FSIG relate to these approaches? Can FSIG give adequate or appropriate syntactic descriptions to natural language sentences or is it limited to approximations of context-free equivalent grammars? Can FSIG contribute anything interesting to the state-of-the art with this respect?

Adequacy of Structural Descriptions What do I mean by adequacy or appropriateness of structural descriptions? Obviously adequacy must be judged by some external criteria. Joshi et al. (2000) maintain that adequate structural descriptions should be semantically coherent. Accordingly, syntactic analysis should be compared against the semantic structure. The idea of coherence is taken even further in the tradition of combinatorial categorial grammars, where the syntax is seen as an artefact rather than an independent level of representation: the syntactic structure is build as the result of

inferences over categories, but these categories are only reflections of the underlying meaning.

Adequacy of classes of grammars has been usually discussed in terms of two formal properties: WGP and SGP. Such measures tend to bias the discussion towards CH and other properties of the set described. However, sematic coherence is not measured directly by WGP or SGP.

Becker et al. (1992) introduced a third way of arguing about the semantic coherence and the adequacy of formal systems. They call this measure the *derivational generative power* (DGP). They motivate it as follows:

> *The starting point for our definition is the observation that while the specifics of a syntactic analysis of a particular construction may be subject to controversy, the (semantic) predicate-argument structure of sentences is fairly uncontroversial. Furthermore, what characterizes many syntactic phenomena in natural languages is the way in which the predicate-argument structures map onto the surface strings.*

According to this view, semantical coherence can be understood as the simplicity of the transformation between structural descriptions and correct predicate-argument structures.

Dependency syntax (Tesnière, 1969 (1959)) is a syntactic framework that captures the predicate-argument structures.

Approximated dependency syntax Our problem is to investigate whether star-free FSIGs could be used successfully (i) as a performance-motivated grammar for *linguistically adequate sets of dependency structures*, and (ii) for representation of parameterized subsets of the SGP of mildly context sensitive grammars. This problem is difficult because the star-free FSIGs, a framework for star-free sets of grammatical model strings and regular surface languages, is not mildly context sensitive, although it might allow approximations for such grammars.

4.1.3 Articles

I have investigated the stated problem by examining dependency structures in a treebank and then building a formal account that is suited for FSIG approximations. The investigations cover, basically, the following three aspects of the problem:

- a measure for complexity of used syntactic structures [4]
- the design of the underlying formal grammar [5]
- an FSIG approach to non-projective dependency grammars [6].

These aspects have been investigated in three contributions in the following way:

Measuring the Syntactic Complexity of Linguistic Data

> [4] Yli-Jyrä, 2003c, "Multiplanarity — a model for dependency structures in treebanks"

This article presents a model that measures the complexity of dependency trees. Then the author conjectures that dependency trees that are highly complex according to the measure are either very rare, incorrect or fall outside of the performance of language users. The measure is tested against an existing treebank that contains dependency trees for a few thousand Scandinavian (Danish) sentences.

Designing the Underlying Formal Model

> [5] Yli-Jyrä and Nykänen, 2004, "A Hierarchy of mildly context-sensitive dependency grammars"

This article elaborates the core of the model sketched in [4]. The resulting computational model, *colored multiplanar link grammar* (CMLG), is designed here so that it is *mildly context sensitive* (MCS) but describes dependency structures rather than constituency structures. This aspect makes CMLG linguistically interesting: it might be able to give adequate structural descriptions to natural language sentences.

Another class of dependency grammars, *colored non-projective dependency grammar* (CNDGs), enforces heuristic acyclicity and treeness conditions on dependency structures. The motivation for heuristics is that acyclicity of the derived structures is not a consequence of the tree-shaped (in fact a string-shaped) derivation tree.

Defining an FSIG Approximation

> [6] Yli-Jyrä, 2004a, "Axiomatization of restricted non-projective dependency trees through finite-state constraints that analyze crossing bracketings"

This article is a dense description of a new FSIG type that can be used to approximate CNDGs. The deep strings of this FSIG are particularly designed to encode CNDGs derivations — as well as the derived strings[1]. Because the essential properties of valid deep strings are difficult to describe without access to the original CNDG, I am forced to specify their properties through complex axioms each of which is defined both verbosely and using a regular expressions as the meta language.

4.1.4 Outline

The three articles correspond to respective sections 4.2 - 4.4 where they are discussed in more detail. This chapter is concluded by section 4.5.

[1] A peculiarity of CNDG is that the strings are derived from left to right and it is the link structure that is interesting from the syntax point of view.

4.2 A Measure for Syntactic Complexity

Article

> "Multiplanarity — a model for dependency structures in treebanks," in *TLT 2003*.

4.2.1 Introduction

Summary

This article sketches a model that is is capable of encoding all dependency structures of a dependency treebank. The amount of resources consumed by the encoding defines a complexity measure for dependency structures.

The article – despite some of its technical confusions that are commented on in Appendix A – contains a number of important innovations. These innovations have already given rise to several follow-up publications where these innovations have been used and developed further (Yli-Jyrä, 2004b, 2005a, 2004a; Yli-Jyrä and Nykänen, 2004).

Motivations

The main limitation of earlier FSIGs has been the representation of syntactic analyses. Due to this limitation FSIG has been regarded as a surface syntactic approach.

An important requirement for a less limited FSIG representation is to keep the string alphabet fixed in the representation. Due to this requirement, a representation or encoding whose alphabet size grows according to the size of the encoded structure is not appropriate.

Definitions

Treebank is a linguistic corpus of sentences with attached parse trees.

The term *tree* is ambiguous. A warning on conceptual confusion is perhaps required: A so-called dependency tree in linguistic analysis can be in fact mathematically a non-tree or at least ordered dependency tree, if we take the linear precedence relation associated with the dependency tree nodes into consideration. Even if we add so-called secondary links to dependency trees, linguists still often talk about trees.

The article talks about planarity, but a better term would have been semi-planarity, because each plane can actually contain two semi-planar graphs: one which is drawn above the sentence and the one that is drawn below. However, the current choice in the article is motivated because it is more convenient to talk about multi-planarity than multi-semi-planarity.

4.2.2 Main Results

(I) A String Representation for General Dependency Graphs

The article generalizes the idea of planar graphs to non-planar as follows: A non-planar graph can be represented as a union of semi-planar subgraphs. Each semi-planar dependency graph can be represented as a bracketed string.

A Remark This is a generalization of the dependency bracketing scheme that was presented in (Yli-Jyrä, 2005a).

(II) Alignment Constraints

The article introduces plane alignment constraints for multi-planar graphs, which constitute a basis for the other two articles in this group:

- Plane Locking — only the top-most semi-plane is active for adding new edges;
- Left Conjoin — right-ward links of each word are stored to the same semi-plane;
- Continuous Tiling — starting with a higher semi-plane is not allowed unless this avoids a collision with a link stored in the current plane.

(III) A Measure for Structural Complexity

The article defines a measure for structural complexity of dependency graphs. This measure is obtained as follows:

- A non-planar graph can be represented as a union of semi-planar subgraphs.
- Alignment constraints regulate how the non-projective graphs are split into semi-planar subgraphs[2].
- The number of these subgraphs is measured.

A Remark The number of (semi-)planes depends on the presence of the alignment constraints. Without alignment constraints we would not obtain the same measure. If we had wanted we could have abandoned the alignment constraints and searched for the optimal decomposition into semi-planar graphs, but such optimization would have been too expensive.

Although alignment constraints prevent us from using the semi-planes in the optimal way, they also guarantee for each graph a unique way to allocate semi-planes. As a consequence of uniqueness, the constraints also prevent us allocating more planes than are necessary for obtaining this unique representation.

The experiments carried out in this paper suggest that the number of (semi-)planes (or pushdown stacks) needed to capture the examples in the corpus was quite small (between 3 and 5).

[2]The decomposition into subgraphs is unambiguous, and there is a deterministic algorithm that implements the alignment constraints as a function that maps dependency graphs to multi-planar representations (Yli-Jyrä, 2004b).

(IV) Rough Parameters for Danish

For 5540 Danish sentences in the studied treebank (Kromann et al., 2003), I found that a 5538/5540 coverage will be achieved with three planes only (I did not represent secondary links).

A Remark According to M. Kromann (personal communication, 2003), all the complex sentences depicted in the article have been actually correctly analyzed in the treebank. This indicates, on one hand, that even the complex sets of structures can be real sentences, and on the other hand, that the model could not point out errors in the treebank by picking up these distinguished examples — in contrary to our expectations. The model might detect errors, however, in some mode dynamic setting, *e.g.* during the annotation process. Alternatively, the complexity measure can be used for automatic ranking of alternative analyses.

At TLT 2003, I was asked for the actual distribution of sentences with different numbers of semi-planes. Afterwards, I carried out more measurements and embedded a short answer to the inquiry inside a popularizing article (Yli-Jyrä, 2004b). A further study on the multiplanarity complexity of various languages and dependency treebanks would undoubtedly be a nice follow-up.

4.2.3 Relevance

Linguistic Interpretation of the Measure Setting a particular bound for the number of semi-planes would claim a fixed property in the language use, *i.e.* performance. Could such a bound also be a competence property, *i.e.* characterized by a naturally class of grammars?

The following points suggest, in fact, that a restriction on the number of scrambled items (which corresponds to the restriction on the number of semi-planes) is more naturally a competence feature than *e.g.* a limit for clausal center-embedding.

First, Joshi et al. (2000) argued that a low bound for scrambling complexity could be motivated as a competence property rather than seeing it as a performance property as is usually done. However, the various *tree-adjoining grammar*s (TAGs) that Joshi has in his mind do not have a parameter such as the number of available semi-planes, unless multi-component TAGs are not taken into consideration. Accordingly, Joshi's argument does not necessarily apply to the measure that I have in our mind.

Second, Rogers (2003, 2004) argues that in mildly context-sensitive tree sets the competence and performance could in a certain sense coincide

> *Here [in multidimensional trees] we have additional flexibility. In choosing the level of the competence grammar in the multi-dimensional hierarchy, we set the boundary on the complexity of the scrambling we admit. On the other hand, given that the level of the grammar corresponds to the number of hierarchical relations we use in encoding the structure of the utterances, one could make a plausible argument that the level of the grammar might be determined by performance considerations, such as working memory limitations. In this way one might arrive at an account*

> *of the limits on the complexity of scrambling that was simultaneously performance based — a consequence of bounds on working memory — and competence based — a consequence of the complexity of the grammars which can be processed within those bounds.*

A similar approach has been presented by Kallmeyer and Yoon (2004).

Third, there exists another account of scrambling where performance limits and classes of dependency grammars are related to each other (Holan et al., 1998; Holan, 2002).

However, our hierarchy differs crucially from Roger's approach: we do not relate semi-planes to hierarchical relations (of different sorts) used in encoding the structure of utterances. Instead, the planes are determined purely on the basis of the unlabeled structure. The alignment constraints are completely blind with respect to the types of different grammatical relations, and it seems that coupling selectivity within this respect would severely reduce the coverage unless the alignment constraints – and unambiguity – are not relaxed.

Connection to FSIG In the article (Yli-Jyrä, 2003c, p.198), we suggest that the presented model lends itself to finite-state approximations:

> *Our model suggests a basis for the complexity hierarchy of dependency grammars, and motivates a class of efficient parsers that cope with mildly context-sensitive sets of dependency trees. Finally, it seems feasible that such a parser can be easily approximated using finite-state methods.*

This suggestion has actually been realized in our later writings: (i) the complexity hierarchy was presented in (Yli-Jyrä and Nykänen, 2004) and (ii) a finite-state approximation of the hierarchy was presented in (Yli-Jyrä, 2004a).

4.2.4 Other Results

A Suggestion for a Heuristic Acyclicity Testing In addition to the multi-planar model, the article suggests a heuristic method for determining acyclicity of dependency graphs. The method involves the so-called Cycle Cutting constraint that was meant to rule out all cycles (at the expense of losing some acyclic structures as well). Such a heuristic test would be useful when implementing a polynomial parser, because with it we could avoid defining acyclicity in general (it would be difficult because acyclicity is not definable in first-order logic (Immerman, 1999)).

The article presented an *ad hoc* method for acyclicity testing. The method did not exploit multi-planar representation of dependency graphs. In a subsequent paper (Yli-Jyrä and Nykänen, 2004), we abandoned this method and developed another that is more tightly coupled with the multi-planar representation of the graphs and the alignment constraints. The third paper (Yli-Jyrä, 2004a) made an attempt to generalize this method even more, by taking advantage of articulation nodes.

4.3 Design of the Underlying Formal Model

Article

> "A Hierarchy of mildly context-sensitive dependency grammars," in *Proceedings of Formal Grammar 2004, Nancy*.

In this article, the contributions of the two authors could be distinguished as follows: we worked for several days writing definitions together, discussing a lot and developing mathematical basis for the new concepts. The second author worked on section 11.2 and the first author worked mainly on the other sections. The first author suggested several extensions in some definitions in 11.2 as needed, and the second author suggested presentational improvements to some formulas of the first author and did some proof reading. The final version, as it stands, was prepared by the first author during the last few days before its final submission.

4.3.1 Introduction

Summary

The paper proposes new classes of link and dependency grammars.

Motivations

Formal language hierarchies for dependency grammars have not been studied extensively, but researchers are interested to see work on such hierarchies (Kruijff, 2002). The measure presented in (Yli-Jyrä, 2003c) was a promising possibility for defining such hierarchies. The measure and the multi-planar representation were developed because we needed a more general and linguistically more interesting dependency-based grammars in order to obtain better FSIG approximations.

Definitions

Link Grammars (Sleator and Temperley, 1991) present structures as semi-planar graphs. The edges that are incident with a lexical node are specified in lexical entries of Link Grammar. Each lexical entry specifies types and left-right-directions of the incident edges.

A *lexicalized grammar* is usually a grammar whose rules are practically moved to the lexicon by inserting lexical anchors to the rules.

Joshi (1985) defines MCSGs loosely as grammars

- that are polynomially parseable,
- that are capable of describing some limited crossing dependencies,
- that generate a superclass of context-free languages,
- whose languages have the constant (linear) growth property.

This definition covers *e.g.* various classes of TAGs. Such usual requirements as the capability of generating copy languages are secondary properties, and therefore *e.g.* TAG *without local constraints* could be seen as a mildly-context sensitive grammar although it lacks this capability (Joshi 2004, private communication).

Each grammar formalism specifies a *domain of locality* (DL) *i.e.* a domain over which various dependencies can be specified. In a CFG the domain of locality is the one-level tree corresponding to a rule in a CFG. Formalism A is said to provide an *extended domain of locality* (EDL) as compared to a formalism B if there is a linguistic constraint which is not specifiable in the local domains associated with B but which is specifiable in the domains associated with A.

4.3.2 Main Results

(I) A Generalization of Link Grammars

The paper presents CMLGs, a generalization of Link Grammars. CMLGs presents structures in a way that is similar to the multiplanarity with alignment constraints (Yli-Jyrä, 2003c).

(II) A Parameterized Acyclicity Condition

The paper presents new heuristics that can be used to enforce acyclicity in the graphs, if the links are directed (from governor to dependent). This heuristics is based on a measure of non-projectivity[3] depth. Non-projectivity depth measures, roughly, how many times an incoming edge of a node is embraced by an outgoing edge when the paths from the root are followed to the leaves. The grammar implements this acyclicity condition by measuring the non-projectivity depths of all nodes up to a parameter-specified bound.

Remarks It is probable that a very small bound for the non-projectivity depth is sufficient for natural languages.

The non-projectivity depth, as presented in the paper, is not cleared (set to zero) at the "articulation" nodes. "Articulation" nodes are words that are not crossed by any dependency links and for which all the in-coming arcs are on opposite sides than the out-going arcs. Clearing the non-projectivity depth of out-going arcs at such articulation nodes would be a nice technique to reduce the number of acyclic graphs that now have been discarded. In the companion paper (Yli-Jyrä, 2004a), the articulation nodes have been processed in this more optimal way.

(III) Hierarchies for These Mildly Context-Sensitive Grammars

The article introduces two parameters – *nested crossing depth* and *non-projectivity depth* — that define classes of mildly context-sensitive link grammars and dependency

[3]The name *non-projectivity depth* for the measure is particularly unmotivated. There are non-projective trees for which this measure gets value 0 (see figure 7 of Marcus, 1967). Obviously, a better name for our measure would be needed. Note that this measure is a generalization of axiom 12 of (Yli-Jyrä, 2005a).

	CNDGs Directed Acyclic Graphs			CMLGs Directed Graphs
$\vdots$				
$c = 3$	$\ni a^n b^n c^n d^n$			
$c = 2$	$\ni a^n b^n c^n$			
$c = 1$	$\supset$ HGDG			LinkG
	$t = 0$	$t = 1$	$\ldots$	

Table 4.1: The hierarchies for CNDGs and CMLGs.

grammars in two different dimensions. The hierarchy based on nested crossing depth (the number of colors needed) had already been implicitly present in my experimental paper (Yli-Jyrä, 2003c). The non-projectivity depth induces a new hierarchy that had not been presented earlier.

Illustration The obtained hierarchies are illustrated in figure 4.1. Identifying the first level of CMLGs-hierarchy to the Link Grammar (depicted as LinkG) requires a certain simplistic notion of Link Grammars that differs slightly from the really implemented Link Grammars for English.

A Remark Some mildly context-sensitive hierarchies for certain *linear context-free rewriting system*s (LCFRSs) exist (*e.g.* Weir, 1992; Wartena, 2001; Rogers, 2003).

4.3.3 Other Results

(I) Extending the Domain of Locality in the Underlying Grammar

The new grammar systems, CMLG and CNDG, have been implemented in the paper, by conversion to an underlying grammars whose technical name is *context-free linear storage grammar with extended domain of locality* (CF-L-EDL-$\mathcal{S}$-G). This class generalizes the class of grammars that has been defined by Wartena (2001) by extending its domain of locality to cover all dependency links that are needed when one word is processed.

A Remark The approach called *complicate locally, simplify globally* (CLSG) (Joshi, 2004) pushes non-local dependencies into the local neighborhood. The goal of the CLSG approach is to find a formalism that would provide local domains large enough so that, in principle, all linguistic constraints (pieces of linguistic theory) such as predicate-argument structures could be specified over these bounded local domains.

The new grammar systems presented in the paper seems to implement the CLSG approach with respect to the predicate-argument structures. We can also say that the

derivational generative power of our new grammars is sufficient for exactly expressing the dependency links of any finite set of dependency trees (obviously a CNDG grammar fitted to a treebank would also generalize and describe some unseen trees as well).

However, there may be other linguistic constraints that are not expressible by means of dependency links. Such constraints are mainly related to the description of word order. Word order constraints in particular are one of the most difficult areas in dependency syntax, as well as in CNDG grammars.

(II) Include Alignment Constraints to Storage Grammars

The alignment constraints guarantee an unambiguous derivation for each dependency tree. The current paper formalizes these constraints and shows that they are reducible to CF-L-EDL-*S*-G productions. It also suggests a new storage type, *normalized storage*, that encapsules this reduction.

4.4 An FSIG Approach to Non-Projectivity

Article

> "Axiomatization of restricted non-projective dependency trees through finite-state constraints that analyze crossing bracketings," in *COLING DG Workshop 2004*.

4.4.1 Introduction

Summary

This article presents a regular approximation for CNDGs. The approximation is given in the form of an FSIG, where a number of parameters determine the level of complexity captured in dependency structures. A *reduced bracketing* scheme for bracketed dependency structures was sketched.

Motivations

As I already axiomatized HGDGs (Yli-Jyrä, 2005a)[4], the next challenge suggested itself: can I extend the approach to CNDGs where we have crossing dependencies? In order to make the extension, several encoding issues have to be solved.

4.4.2 Main Results

(I) String-Encoding of CNDG Derivations

The article demonstrates that all the information that is needed to represent derivations of CNDG can be encoded into annotated surface strings. It is particularly nice to

[4] This paper was written first.

see that alignment constraints (Yli-Jyrä, 2003c) (corresponding to the normalized storage) can be formalized without big difficulties as constraint languages when we use a bracket-based representation for the content of each pushdown.

A Remark The obtained bracket-based representation is very similar to the original sketch (Yli-Jyrä, 2003c): it adds only (i) the wall node, (ii) indication for active color at each position, and (iii) non-projectivity-depth counters. The presentations would have been even more similar if we had not employed our reduced bracketing scheme that caused some complications.

The string representation encodes derivations as a sequence of local, multi-dimensional trees. This resembles the super-tagging approach suggested for TAG parsing, but here each "super-tag" — the subsequence surrounding each word between two word boundaries — consists of smaller pieces that could, in principle, be assigned by separate taggers.

(II) FSIGs and CNDG Can Give Equally Adequate Structural Descriptions if There Are Appropriate Performance Limits

In addition to *nested crossing depth* and *non-projectivity depth* that are defined for CNDG, the FSIG approximation obtained in the article uses two new performance-oriented parameters to restrict the complexity of non-projective dependency trees:

- The first one, *proper embracement depth*, is built on top of the reduced bracketing scheme.
- The second, which does not have any name (in section 5 of the article), is used to restrict the amount of subcategorization information that can be transmitted from string-local domains of words to the actual dependents (and the governors) in the sentence through labels of brackets.

These additional limits provide a way to extract performance-compatible FSIG restrictions from CNDGs.

A Remark Most of the parameters presented measure the amount of information that can be transmitted between two positions through links. This comes very close to the narrowness of short-term memory and to the graph-theoretic complexity measure called *path-width*, which is reflected in some parsing approaches (*e.g.* Kornai and Tuza, 1992) and observed in psycholinguistic experiments (*e.g.* Gibson, 1998).

4.4.3 Linguistic Relevance

Structures Axiomatizable through Their String Encoding The article suggests that very complex graphs such as those generated by mildly context sensitive grammars could be axiomatized through their string encoding. However, further research in this area is needed: it would be tantalizing to find out whether similar representations were available for TAGs or *combinatory categorial grammar*s (CCGs).

In the axiomatization presented in the article, I assumed that certain parameters are fixed. However, if the special languages for balanced bracketing would be idealized to context-free languages — recall that context-free Dyck derivatives were our starting point in two earlier grammar representations (Yli-Jyrä, submitted 2003, 2005a) — then we could get the coverage of CNDGs. However, we did not try explicitly to build a Chomsky-Schützenberger style representation for CNDGs in this article.

One More Star-free Grammar If the axioms were studied more carefully using the methods presented in (Yli-Jyrä, 2003a), it would be relatively easy to show that they define star-free, FO$[<]$ definable regular languages. Of course, studying these directions is not the subject of this article, nor the theme of the current chapter.

Is Embedding More Complex than Crossing? A surprising and possibly very important observation on the structure of the axiom set can be made: although the bracketing with multiple colors simulates multi-pushdown automata, different subsets of the resulting scheme stay relatively independent, as each of them correspond to disjoint sets of axioms.

Although the separation increases the number of axioms, it helps us to see that the dot-depth of the whole grammar is not higher than of any of these sub-grammars. In fact, we would get a lower dot-depth complexity if we could increase the number of colors while reducing the depth of bracketing (we cannot do this as long as we rely on the alignment constraints).

This suggests that our approach to non-projective trees does not make FSIG grammars computationally more complex. The effect could be very well be the opposite. This leaves us with the following challenging questions: Is the *dot-depth hierarchy* (DDH) more robust than the *Chomsky hierarchy* (CH) when performance of natural language is studied? Is multiplanarity (the number of stacks) a minor dimension compared to the dot-depth (the stack size)?

4.4.4 Other Results

(I) Reduced Bracketing Does Not Give Much Advantage

The article contains some indication of possible problems related to the new representation. The problems concern, in particular, reduced bracketing. Reduced bracketing, while useful in some respects, is also a source of additional difficulties: according to section 5, the number of axioms grows linearly to a degree-like bound (r) for the dependency graphs.

A Remark The extraneous axioms could have been avoided if full bracketing had been used. Then, however, the required proper-embracement depth would grow to compensate for the lost advantage of reduced bracketing.

In dependency bracketing, both sides of the brackets need to carry information of the bracket label. The bracket labels are needed on both sides to implement co-occurrence constraints specified for lexical categories. With full bracketing, constraints can be enforced in the local bracket neighborhood of each word.

The effect of reduced bracketing is in strict contrast with approximations of context-free bracketing grammars, where reduced bracketing gives much more than it takes. Similar reduced bracketing would be too unredundant for dependency structures, and it would require inelegant compensation techniques, especially copying of labels. Therefore, reduced bracketing does not seem to be a good idea in representations for dependency structures.

(II) The Bracketed Strings Could Be Processed with a Multi-Tape Model

The introduction of the article presents an annotated surface string (deep string) by partitioning it into subsequences. These substrings are presented as if they were on synchronized tapes. We wrote: "*When these subsequences are put on top of each other we obtain the following combination:...*" This implicitly anticipates a possible implementation of the grammar using multiple, synchronized tapes. In *Future Work* section of the article, a multi-tape automaton is mentioned as a possible framework for combining different kinds of deep strings into the same framework.

A Remark The idea of special synchronized multi-tape automata — or an equivalent model — is elaborated further in chapter 5 of this book.

4.5 Conclusions

In this chapter, I have summarized the results of the three articles mentioned in the beginning of this chapter. These publications investigate how to extend flat representations of dependency structures to cover non-projective dependency trees. The results indicate that the new star-free FSIG framework (Yli-Jyrä, submitted 2003, 2005a) can be extended towards non-projective dependency trees.

As I approach non-projective dependency parsing and description of free word-order languages, I reach a difficult area where many new interesting and open problems start to come up much faster than can be exhausted in this short study.

I have dealt with some of these problems with considerable success and defined same new concepts, grammars, measures, hierarchies and representations that can be applied in the FSIG framework. The main practical outcome is a new type of FSIG. It is probably widely applicable to free-word order languages and potentially efficiently parseable in practice.

The result also reveals many theoretical questions about the representations of mildly context-sensitive grammars and on the computational complexity of their regular, performance-compatible approximations.

Chapter 5

Parsing Strategies

5.1 Orientation

5.1.1 Summary

This chapter basically proposes three optimization techniques for FSIG parsing. These techniques are closely related to some generalized compilation methods for FSIG constraints and they optimize the FSIG parsing through decompositions that reduce the size of the manipulated automata.

5.1.2 Problem

The current chapter investigates some parsing strategies with the aim of reduction the overall parsing time complexity of FSIG parsing.

FSIG parser development is a huge challenge that cannot be solved merely through employment of code-level optimizations in a library of automata operations. During the period 1995 – 2002, I actually did quite a bit low-level programming (more or less as a hobby) with the aim to implement better FSIG parsers. But I achieved only relatively small improvements that are not of interest to the current dissertation.

However, more abstract investigation of parser design provides us with a complementary approach that does not try to implement the whole parser before finding essential strategic improvements. This complementary approach has now turned out to be very fruitful in terms of the gained insight. In order to attack, in the sequel, the grand challenge of FSIG parser development, I will focus on solving three fundamental problems. These problems have to do with the succinctness of finite automata during FSIG parsing and they are the following:

1. state blow-up of the constraint automata as a function of the depth of bracketing,

2. state complexity of the intermediate and final results as a function of the bracketing depth, and

3. state complexity of the intermediate results when a high number of sparse constraints interact at a small number of distant string positions.

5.1.3 Articles

Considering the three succinctness problems mentioned above, I propose, respectively, the following three techniques or strategies:

1. Decomposing the Grammar with Respect to Bracketing Depth,
2. Keeping Sub-grammars in Separation,
3. Solving Hard Portions Separately.

These strategies have been discussed in the contributed articles in the following way:

Decomposing the Grammar with Respect to Bracketing Depth

> [7] Yli-Jyrä and Koskenniemi, 2004, "Compiling contextual restrictions on strings into finite-state automata"

The article contains many new *compilation methods* for restriction operations. It presents, in particular, an efficient solution to a long-standing problem that concerns compilation of *context restriction operation with overlapping centers*. This solution has already been adopted into a proprietary finite-state compiler (XFST), but some of the further methods in the paper are better motivated in the FSIG application context.

In addition to the new compilation methods for context restrictions, the article introduces *generalized restriction*, a conceptually elegant extension of the context restriction operation. The generalized restriction operation can be used in many ways. For example, I can solve with it the problem where some context restrictions grow exponentially according to the depth of bracketing. The growth cannot be avoided as long as the context restriction is represented as a single automaton. The generalization can be used to obtain a *parallel decomposition* of a context restriction. Each slice in the decomposition can be compiled into substantially smaller automata than we would get by compiling the original constraint into a single automaton.

The smaller, decomposed representation helps to reduce the amount of memory that is needed to store constraint automata and their combinations. When the whole grammar is decomposed according to different bracketing levels, we obtain, for each bracketing level, a set of constraints that will be called, in this chapter, a *sub-grammar*. Each sub-grammar contains smaller automata than the original grammar. In contrast to the collection of original, monolithic constraints, sub-grammars have slightly better chances for combining a large number of constraints into a single automaton.

Keeping Sub-grammars in Separation

> [8] Yli-Jyrä, 2004d, "Simplification of Intermediate Results during Intersection of Multiple Weighted Automata."

This article, or actually an extended abstract, sketches a methodological framework for optimizations that are related to computation of intersection of multiple automata[1].

The motivation for the presented approach comes from FSIG parsing. Typically, FSIG parsing admits a quite small result containing only a few alternative parses — especially if the grammar is Eng-FSIG. However, this is not true in the worst case scenarios. With certain pathological grammars such as approximations for FCFBGs and other "*Bracketed finite-state intersection grammar*s (B-FSIGs)", the final result grows, in the worst case, exponentially according to the depth of bracketing realized in the sentence, if the grammar (or its bracketing depth parameter) is part of the input. To avoid exponential state-space blow-up of *intermediate results*, we need to

- compute intersections with the sentence automaton separately for each bracketing level (sub-grammar) to obtain a *compact representation* (that consists of several revised sentence automata), and
- enforce consistency in the compact representation before expansion to the *final intersection result* whose size can be, in the worst case, exponential.

Enforcing consistency between the revised sentence automata can be done through appropriate *structure sharing* techniques.

Implementation of structure sharing between separate automata require efficient simplification methods. The simplifications presented in the article are capable of extracting from automata generalisations that function as shareable interfaces for the original automata. Different pairs of automata give rise to different interfaces between them. Two automata can be made consistent (i) by making their respective interfaces consistent through intersection, (ii) by restoring the hidden letters in the interfaces and (iii) by combining the resulting interface automaton with the original automata.

Solving Hard Portions Separately

[9] Yli-Jyrä, 1995, "Schematic Finite-State Intersection Parsing"

This paper is, as a publication, in a different category than the other eight articles in this thesis. It is, however, the original reference where I first proposed an alphabet extension and minimization method for FSIG. Some restricted methods presented in this paper constituted a methodological background for more general approach that we presented in [8].

In contrast to the new generalized simplification framework [8] that focuses on "vertical" consistency between different levels of bracketing, this older paper discusses

[1] Many automata theorists would see intersection as an operation on languages rather than on automata recognizing them. However, the direct product operation (Rabin and Scott, 1959) is too much associated with certain automaton construction while we would like to stay here at a higher level of abstraction. Recall that for example union of automata can be implemented in many different ways: with or without epsilon transitions, with optional determinization and with optional minimization *etc*. The same holds for intersection of automata that can be computed also using *e.g.* DeMorgan's law *etc*. My early article (Yli-Jyrä, 1995) is, unfortunately, even more confusing as I identified in it the *intersection automaton* with the accessible subset of the direct product.

"horizontal" consistency — long distance dependencies between position of the annotated strings.[2] Long distance constraints constitute a problem for example if the constraints that relate the beginning and the end of the annotated strings are applied to a sentence automaton that has of lot of alternative states in the middle positions of the annotated strings. Such "thickness" in the middle may be a result of an earlier application of constraints to the sentence automaton. Changing the order of constraint application may help in some cases, but better ordering does not necessarily generalize across sentences. However, there are some simplification methods that can extract portions of the sentence automata and admit a much more general solution.

In the article, the idea of alphabet expansion was introduced to the FSIG framework. The expansion was done here on the sentence-by-sentence basis. It makes the sentence automaton and its restrictions — "*specialized rule automata* (SRAs)" — structurally simpler[3]. The alphabet expansion transforms each annotated string in the sentence automaton into a string that represents an unordered set of position-symbol pairs. Although the original linear order is retained in simplifications, absolute positions of the symbols are now less crucial than in the original automaton. This leads to two simplification methods that take advantage of the new alphabet: (i) the folding procedure and (ii) an implicit representation of hidden transitions.

5.1.4 Outline of the Chapter

The substance and relevance of those methods that arise in the three accompanying articles is discussed more broadly in sections 5.2 – 5.4. In section 5.5, I will give a summary of some other approaches to FSIG parsing. These methods will not be considered in my dissertation. The chapter is summarized in section 5.6.

5.2 Decomposing Grammar with Respect to Bracketing Depth

Article

> "Compiling contextual restrictions on strings into finite-state automata", *Proceedings of Eindhoven FASTAR Days*.

The paper contains a remark outlining the relative efforts of each author.

5.2.1 Introduction

Summary The article presents new compilation methods. Some of these can be used to decompose constraints in FSIGs into separate constraints that will compile into smaller automata.

[2] Note that long-distance dependencies in strings do not fully coincide with long-distance dependencies in constituent-based structures.

[3] After the extension, the original sentence automaton recognizes a local string set, and the *compact folded* SRAs represent languages whose structure reminds of piecewise testable languages (Simon, 1975) (*cf.* (Yli-Jyrä, 2001).

Motivations Decomposing FSIG assertions into components according to the bracketing level is expected to have a big impact on the size of each constraint, and the way the constraints in the grammar can be grouped together. All this might contribute to parsing efficiency.

The idea of levels in FSIG originates from Kimmo Koskenniemi (personal communication, 2001), but it has counterparts in other finite-state parsing frameworks (including parsing with cascaded chunk parsers or iterated application of finite-state transducers *c.f.* Roche, 1997).

Previous Work Before we discovered the solution presented in the article, I had made the following findings:

- I introduced a depth parameter that determined how many brackets may be nested in each FSIG (Yli-Jyrä, 2003a). This made it possible to argue about an exponential state complexity with respect to the bracketing depth.
- Different ways to compile bracketing restriction of FCFBGs (Yli-Jyrä, 2003d, submitted 2003) were found[4]. Investigation of their state complexity helped us to understand how the size of automata grows as a function of the depth parameter.
- In (Yli-Jyrä, 2003a) I defined the semantics of general case of the context restriction operation using concatenation and boolean operations. This was a correct method and a precursor for the new, simpler methods presented in (Yli-Jyrä and Koskenniemi, 2004).

5.2.2 Results

(I) A New Compilation Method for Context Restrictions

The article presents a compilation method that has an elegant and simple structure. Due to its efficiency and correctness, the method replaces some previous compilation algorithms in some finite-state tools.

A Remark The compilation method does not need the full implementation of concatenation closures nor transducers. It is a good example of the use of finite number of mark symbols as well as simple homomorphisms and Boolean combinations, in contrast to the iterated marking that is used in many regular operations in natural language processing.

(II) Generalized Restriction Was Introduced

The article shows that many different kinds of restrictions can be seen as special cases of a relatively simple operation, generalized restriction.

[4]One more compilation method was used in a toy grammar that has been put available on the web at `http://www.ling.helsinki.fi/~aylijyra/BracketingCFGs/`.

A Remark Generalized restriction operation opens new possibilities for the description of discontinuous structures such as idioms. Furthermore, using it with bracketed string sets might allow for capturing tree adjoining grammars (*c.f.* Roche, 1997).

(III) Decomposition of Context Restrictions

The paper presents a flexible and correct method for decomposing context restriction constraints into separate constraints according to the bracketing level. Decomposition can make a DFA-based implementation of individual constraints exponentially smaller according to the bound for bracketing depth.

A Remark Decomposition does not require any changes to the parsing algorithm, but it takes one constraint and replaces it with a number of sub-constraints. The intersection of the sub-constraints will return the original one. However, the resulting sub-constraints can be grouped and combined in new ways, which potentially improves the efficiency of applying the constraints to the sentence automaton as the constraints can be synthesized with each other earlier than otherwise.

The parsing can be split into well motivated subproblems and the obtained results can then be combined with each other. For example, it is often easier to construct an algorithm that checks several assertions at a certain level of bracketing than to construct an automaton that checks one assertion at several levels of bracketing (*c.f.* Yli-Jyrä, 2004a).

A Related Open Problem It is, in fact, an interesting open problem whether all sub-constraints affecting the same bracketing level could be combined with each other. This would allow the efficient processing of a chain of bracket pairs at the same nesting level by one pass of an automaton. At the same time, constraints on reduced bracketing at the same level could be checked.

Explanation Decomposing an FSIG constraint set with respect to bracketing depth means splitting the whole grammar into sub-grammars such as “the grammar for sentence level clauses”, “the grammar for the embedded clauses”, “grammar for the double-embedded clauses”, *etc*. In a subgrammar corresponding to a particular bracketing level, the new assertions will not interfere with other levels. The only exception is presented by those context conditions that are dependent on bracketing levels other than the primary level of the subgrammar they belong to. Such context conditions are handled correctly with our decomposition method, which is an important achievement and necessary for implementing the original idea of the second author correctly.

Illustration: Fully Specified Tape Figure 5.1 shows a string or an input tape of a one-tape sentence automaton. This annotated string is used to represent a dependency tree in (Yli-Jyrä, 2005a).

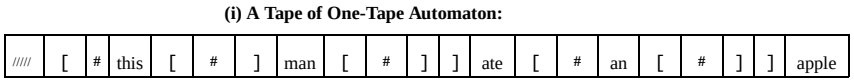


Figure 5.1: A single tape containing a bracketed dependency tree.

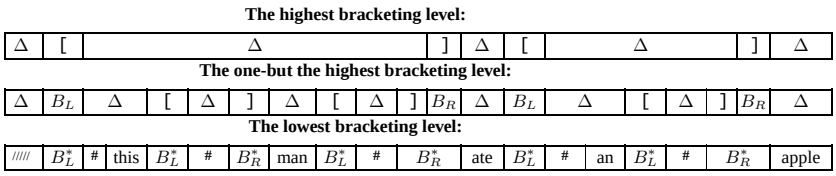


Figure 5.2: Decomposing a string into three languages.

Illustration: Under-Specific Tapes Figure 5.2 illustrates the effect of decomposition of constraints in a very informal and rough way.

In the decomposition each tape becomes a language that is obtained by abstracting away all the bracketing levels that are not to be specified exactly on the level to which the tape corresponds. The outer brackets are specified by B_L and B_R and the embedded bracketing is depicted with symbol Δ. This symbol, Δ, stands for the set of all substrings with balanced bracketing up to a fixed depth. The last tape represents the surface string as well as under-specified slots from which a specific bracketing can be drawn (B_L^* and B_R^* stand for any sequences of left or right brackets). Note that the intersection of the tapes in figure 5.2 results in the tape in figure 5.1.

In this illustration, we see that each sub-grammar contains full bracketing, but only one level of bracketing is fully specified. In addition to the fully specified level, there are some information on the other bracketing levels.

5.3 Keeping Sub-grammars in Separation

Article

> "Simplification of intermediate results during intersection of multiple weighted automata," in *WATA 2004* (Yli-Jyrä, 2004d).

The algorithms promised in the abstract have been initially implemented, but the code is not included in this dissertation.

5.3.1 Introduction

Summary

The extended abstract sketches a framework for local simplifications of automata during FSIG parsing. The proposed simplifications allow for the building of a consistent compact representation for the ambiguity in the sentence and help avoid *inside* the

parser the kind of state-space explosion that would be possible if the ambiguity was presented using a single DFA as usual.

Problem

A first part of the problem is that decomposition of the grammar does not make the final result of the parser smaller. Sub-grammars created in section 5.2 can be used to do partial parsing in separate sub-parsers. These sub-parsers would apply a sub-grammar to their own copies of the sentence automaton. The final versions of sentence automata in different sub-parsers will represent partial forests only. The whole parser needs to enforce consistency between the versions of the sentence automaton. A naive way is to combine the final versions of the sentence automaton by direct product. A better way would be to enforce consistency between the versions of sentence automata and to produce, as a result, a set of *mutually consistent* sentence automata. The strings common to all these sentence automata can be found easily with a backtrack-free search that starts from the left "corner" of the bracketed strings.

Another part of the problem is that DFA are, as representations of ambiguity, less succinct than *e.g.* non-self-embedding context-free grammars (*c.f.* Anselmo et al., 2003).

The following example shows a pathological FSIG grammar that would make an exponential state-space blow-up when the depth of bracketing is increased:

$$\begin{aligned}
\#__\# &\Rightarrow [_A\Delta]_A|[_B\Delta]_B;\\
[_A__]_A &\Rightarrow a|[_A\Delta]_A|[_B\Delta]_B;\\
[_B__]_B &\Rightarrow a|[_A\Delta]_A|[_B\Delta]_B;
\end{aligned}$$

Assume that Δ is a regular approximation for the set of bracketed string with maximum bracket depth k. When these constraints are compiled and intersected with the language $\{[_A,[_B,]_A,]_B\}^*a\{[_A,[_B,]_A,]_B\}^*$, we obtain 2^k different bracketed strings. The result will require $O(2^k)$ states.

Pathological grammars are relevant objects of consideration because they can lead to substantial improvements that are advantageous to the parser's overall efficiency (*c.f.* Maxwell and Kaplan, 1996). If bracketing depth was still fixed to 1, as in the original Eng-FSIG, we could not find an interesting worst-case scenario with exponential blow-up.

Approach

We can recognize some possible cures for the ambiguity representation problem by considering how FSIG represents its parse forests. While Eng-FSIG is very difficult to analyze due to its flat structure, a more promising observation can be made from our new classes of FSIGs. The new classes of FSIGs make a heavy use of *bracketing* and thus there is an unexploited analogy to non-finite-state grammars and parsers. It is important that FSIG parsing algorithms also make use of elementary trees, even if they are encoded through bracketing.

In many non-regular grammars, elementary trees have a three-fold function:

- Elementary trees constitute the basis for *ambiguity packing and structure sharing* in compact representations of parse forests.

- Co-occurrence constraints can be expressed in the *domain of locality of elementary trees.*

- Parsing algorithms can make use of locality of elementary trees when they enforce *consistency* of parse forests.

When these ideas are transformed to the FSIG, we need, in particular, a better method for structure sharing. (Ambiguity packing for alternative sub-trees is already implementable by minimal automata.) For parsing purposes, FSIG grammars must be transformed to a format that allows efficient consistency enforcing techniques between partial parse forests that are obtained by parsing the input with sub-grammars.

This format should be based on the notion of elementary constituent or dependency trees, although each phrase or dependency link will be represented in FSIG through matching brackets.

5.3.2 Proposed Solution

The extended abstract does not describe the proposed solution in detail. In order to interpret the abstract properly, I have to recapitulate and illustrate the crucial aspects in the proposed solution.

(I) Expanding the Common Alphabet of Automata Enables Referring to the States of the Reference Automaton

The abstract says "we expand the common alphabet of the automata in such a way that it is possible to determine the states of the reference automaton". Alphabet extension gives a simple technique through which the reference automaton can share its state space with other automata.

An Illustration: The Reference Automaton According to the article we need also a reference automaton. In our example, the reference automaton is as in figure 5.3. Each transition that changes state in this automaton has a unique label.

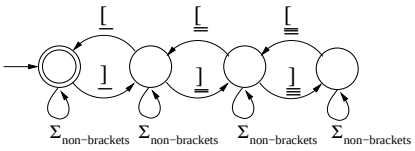


Figure 5.3: The reference automaton.

An Illustration: Annotated String with Extended Alphabet When the alphabet of the grammar is expanded according to correspond to the reference automaton (this expansion is a regular relation), we obtain a new one-tape representation where the brackets have been differentiated according to their level of nesting. To illustrate this, we expanded the alphabet of the input tape in figure 5.1. The tape obtained is depicted in figure 5.4.

A Remark Note that the brackets in the expanded alphabet indicate the source state of the reference automaton (or, in fact, both).

(II) Implementing Projections by Homomorphisms

Simplifications (such as merging of states and substitution of letters with the empty string) resemble "the projection operation of relation tables that is used in query optimization in modern database systems".

The abstract proposes a set of special simplifications during pairwise products of automata. Some of the simplifications would change letters that are ignored by one input automaton to empty strings. This makes the strings shorter, which often leads (through epsilon removal and minimization) to smaller input automata. By means of the following illustration we see how hiding of letters can be used to extract simpler languages.

An Illustration

Projections in the Simplification Framework The proposed simplification methods can be used to extract projections of the one-tape automaton that was shown in figure 5.4. In the simplification framework, a projection operation could be a string homomorphism that preserves the symbols that belong to some specified equivalence classes and maps the others into the empty string. Such an operation corresponds to the simplification that substitutes some letters with the empty string.

The equivalence classes of alphabet of the shown reference automaton are: {[}, {]}, {[}, {]}, {[}, {]}, and {////, #,this,man,ate,an,apple,...}.

We can use projections to extract simpler tapes from the tape shown in figure 5.4. Some resulting projections are shown in figure 5.5.

Simple Multi-Tape Model We need some grounding for the terminological choice *projection* when we talk about extracted tapes. Projection often refers to dropping of attributes or tapes, but is also used for operations that return subintervals (Bowman and Thompson, 2000). In our case we actually might have both the interpretations.

////	[	#	this	[	#	]	man	[	#	]	]	ate	[	#	an	[	#	]	]	apple

Figure 5.4: A single tape with an expanded alphabet.

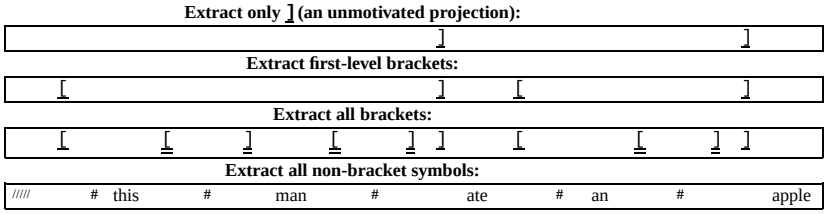


Figure 5.5: The projection operation: some extracted tapes.

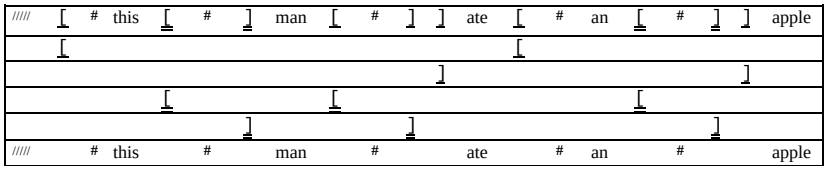


Figure 5.6: A run of a simple multi-tape automaton.

We can actually talk about dropping of tapes. To illustrate this, we interpret the single tape depicted in figure 5.4 as a combination of tapes processed by a multi-tape automaton. Let us first define a simplified model of multi-tape automata as follows:

- We call one of the tapes (tape 0) the *first tape* and the other tapes are called *additional tapes*.
- The first tape is *partitioned* to the other tapes so that each input symbol on tape 0 has a unique copy on some of the other tapes.
- The symbols that are stored to the additional tapes are aligned according their corresponding position in the first tape. The empty space (a place for a cell) stand for an empty string in the input. When one of the additional tapes is being read, the respective head skips automatically to the next symbol.
- The tapes are synchronized. Reading operations on the first tape reads also the corresponding copy from an additional tape. Reading operations on an additional tape also reads the symbol from the first tape. Reading beyond the end of a tape is not permitted.

Figure 5.6 illustrates how the tapes operated by a *simple multi-tape automaton* (SMTA) may be filled. The figure represents tapes in a 4-tape automaton of the proposed special type.

Figure 5.4 showed an example of an automaton with an extended alphabet. This one-tape automaton is also an alternative representation for the SMTA of figure 5.6. Thanks to alphabet extension, a tuple of tapes that are accepted by a SMTA can be represented by means of a string in a one-tape automaton. When such one-tape automata are interpreted as SMTAs, we gain access to crucial notions that are normally available only in the SMTA model.

The states of the reference automaton (figure 5.3) are encoded implicitly by the letter equivalence classes, and now those classes correspond also to the additional tapes of the SMTA.

A Remark The idea about "synchronized" SMTA has some earlier applications in computational linguistics. The KIMMO model or so-called (original) two-level model of morphology (Koskenniemi, 1983) implemented a special class of finite-state transducers as finite-state automata. These transducers specified same length relations (Kaplan and Kay, 1994). More general, but also "synchronized" transducers and SMTAs have been studied in partition-based morphology (see *e.g.* Kiraz, 1994, 1996b,a, 1997, 2000; Kiraz and Grimley-Evans, 1998). Some other possible ways to interpret one-tape automata as SMTAs are very powerful (Tamm et al., 2004; Tamm, 2004).

It is important that the projections contain enough information for recovering the full, synchronized structure when put together. The author is aware of these problems that arise when projections are combined, but the current article does not reveal sufficient conditions for resynchronization of projections. In relational databases, we need to share key attributes while here we have to keep some anchors.

Relational Projections The projection operation is used in relational database systems to optimize join queries (Ullman, 1988). Especially multi-way join queries are often optimized by computing first auxiliary tables where only the common attributes of each pair of tables are joined.

A projection of an SMTA is an SMTA that specifies only a subset of the original tapes. The *projection* operation has been recently defined for a more general multi-tape automaton model (Kempe and Champarnaud, 2004; Kempe et al., 2004a,b).

(III) The Simplification Method Implements Structure Sharing

According to the article (the extended abstract), simplifications make it possible to represent the language of the final sentence automaton as an "intersection" of separate [sentence] automata that correspond to the decomposition obtained in (Yli-Jyrä and Koskenniemi, 2004). According to the this article, simplification method will implement "a kind of structure sharing".

Structure sharing in the relational context could perhaps be understood as follows: a cross-product or a join of two *sets* of structures is represented implicitly or lazily in its decomposed form, without actually expanding the representation into its normal representation. Structure sharing is enabled by a mechanism that connects the sets in an efficient manner. The question is: can we implement join for projections of SMTAs?

The article gives an answer: we can use intersection for the compact representation. This requires, however, that the intersected automata have identical tape alphabets. Fortunately, trivial cycles on the missing letters can be added to the one-tape automaton, as proposed in the article. With this cheap technique the letters that have been removed in earlier projections can be re-introduced to the automaton. The obtained result automaton can now be intersected with the other automata.

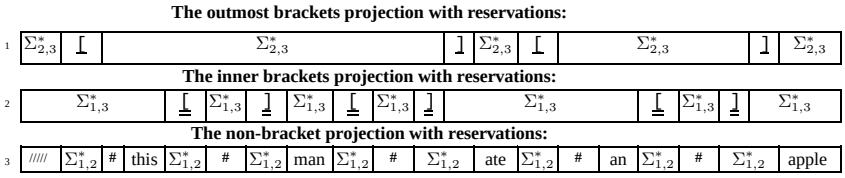


Figure 5.7: Missing letters are inserted to projections during join.

Illustration Figure 5.7 illustrates a few possible tape sets of one-tape automata that are obtained by introduction of the missing letters. These tape sets can be intersected (joined) with the tape set of the fully specified one-level automata such as the one in figure 5.4.

A Remark The simplification method proposed in the abstract (and in its interpretation as multi-tape decompositions) are a special kind of parallel representations of regular languages. The underlying idea is not, however, completely new, because parallel and serial decompositions of finite-state machines have been known for almost fifty years, and a lot has been written about cascades of transducers (and sequential machines) as well.

Relevance

Given the Boolean operations, projection and the join operation that is implemented through re-introduction of letters, we now have a pretty good relational calculus that is implemented by means of one-tape automata, but can also be interpreted in terms of SMTA. The details of this calculus go beyond the contributed abstract.

5.4 Solving Hard Portions Separately

5.4.1 Article

> "Schematic finite-state intersection parsing," a short paper presented in *NODALIDA 1995* (Yli-Jyrä, 1995).

A short comment on this article is perhaps needed. The article is not fully comparable with the other articles included to this dissertation. The article is composed in an admittedly naive style, using some confusing terminological choices that doubtlessly reflect my limitations at the time, 1995, when the article was written. For the time being, the article is, however, the latest and the only available description of some techniques that are part of the methodological framework of my later publication (Yli-Jyrä, 2004d)[5].

[5] I have described these methods more comprehensively in my master's thesis (Yli-Jyrä, 1997), but the thesis has been written in Finnish.

As far as we are concerned with the techniques presented in the article, we can still say that the article presents fundamental insight that led to our further methods. Unfortunately, the article's connection to minimization of incompletely specified asynchronous automata became obvious as late as in summer 2004, and these connections are now an area of further research (Yli-Jyrä, 2004c).

5.4.2 Introduction

Problem

Decomposing the grammar according to the bracketing level helps only if the complexity of the grammar is mainly caused by deep bracketing. The sub-grammars obtained by decomposition may still have a large number of complex constraints.

Moreover, the length of the deep strings processed by the parser grammars are a multiple of the length of the input sentence, and the string may contain as many as a dozen or twenty multi-character symbols per word. When constraint automata are being applied to the sentence automaton, the size of the sentence automaton can grow exponentially as a function of the sentence length, although such a growth will be limited by a linear function that is determined by the immense state complexity of the grammar (Tapanainen, 1997). Accordingly, long sentences can be, in practice, substantially more difficult to parse (with unoptimized algorithms) than short sentences, although we use finite-state techniques.

It is, however, interesting that most constraints automata in the compiled FSIG ignore (*i.e.* never change state on) many letters in the deep strings. Usually they enforce constraints that deal only with some tiny details of the whole string[6]. Therefore, a lot of extra work is spent when the ignored portions of the deep strings are carried forward by constructing new states and transitions for them while direct products are being computed.

Definitions

In the paper, SRAs refers to temporary constraint automata that are computed during parsing from the sentence automaton and a constraint automaton that is in the grammar.

5.4.3 Results

(I) Admissible Languages of Automata

The article proposes keeping the original sentence automaton fixed (as a reference automaton) during parsing.

A Remark In (Yli-Jyrä, 2004d) I introduced a notion of reference automata as a basis for (i) alphabet extensions and (ii) automata simplifications. Because intersection

[6] This is partly a consequence of the preferred tendency towards parsimonious analysis and elegancy in grammar writing.

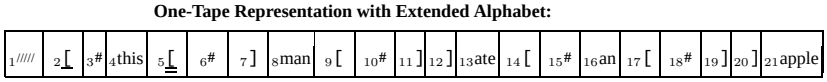


Figure 5.8: One-tape automaton with a horizontally extended alphabet.

(identity mappings) is a special case of composition of relations, we can view a sequence of pairwise intersections as a transducer cascade. In transducer cascades, the output language of previous transducer is the admissible language of the next transducer. The finite-state automata for which the admissible language is given can be incompletely specified, which means that target states of certain transitions in the automaton are not specified. The unspecified target state of an transition can be interpreted as a set of states one of which will be chosen by an implementation (the choice can be fixed or randomly changing).

(II) Incompletely Specified Automata

The article proposes adding some failure transitions to an incomplete automaton. The transitions that were not added were not specified at all, which is in contrast to the usual interpretation of state diagrams. According to the usual interpretation, the invisible transitions correspond to transitions to a useless state.

A Remark For incompletely specified finite automata these there exists a large body of literature on minimization algorithms (the fundamental papers include Paull and Unger, 1959; Grasselli and Luccio, 1965). The problem of *incomplete finite-state machine* (IFSM) minimization is in NP and NP-hard (Pfleeger, 1973; Garey and Johnson, 1979), but some special classes of IFSM have efficient minimization algorithms (Huffman, 1954; McCluskey, 1962; Paull and Waldbaum, 1967; Pager, 1971; Ehrich, 1972; Tomescu, 1972). These classes include the incompletely specified asynchronous sequential machines, which are closely related to the compact folded SRAs.

(III) Simplification of Specialized Rule Automata

The basic technique (Yli-Jyrä, 1995) allows for simplifying SRA. It consists of the following steps:

1. Expansion of the alphabet of the SRA (Yli-Jyrä, 1995). An example is shown in figure 5.8 that illustrates the effect of horizontal alphabet extension.

2. Setting the language of the reference automaton to an input restriction for the SRA.

3. Make the paths in the SRA shorter. This can be done through *folding*[7] reductions (similar to minimization) that do not change the intersection of the reference automaton and the SRA (Yli-Jyrä, 1995).

[7] The use of this term in the article was new rather than previously defined.

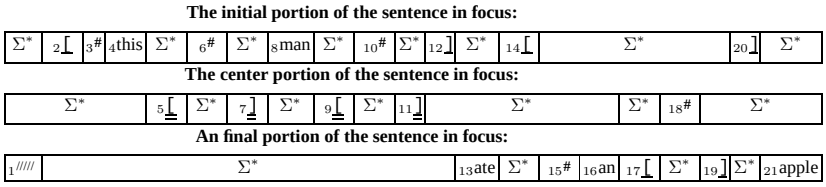


Figure 5.9: Extracting horizontally defined portions of the sentence.

Several modifications to the basic scenario can be made. For example, we can make simplifications before the SRA has been constructed (or during its construction) (Yli-Jyrä, 1997). Furthermore, in our new FSIG grammars the sentence automaton can contain trivial cycles (cycles of length 1), which requires slight modifications to these algorithms.

A Generalization: Projections In addition to these we can include projection into the set of available operations. Projections could be used to extract sub-constraints over a subset of the expanded alphabet. The resulting sub-constraints will ignore the remaining part of the expanded alphabet. (An efficient projection method for sentence automaton was presented by Yli-Jyrä, 1997, and a similar method could be developed for extended alphabet.)

Some examples of what can be done with projections are shown in figure 5.9. The extension allows selecting symbols of interest into projections where other symbols are simply ignored. Under certain assumptions we can actually allow *any* symbol to occur, thus we use Σ^*, the universal language, in the *don't care* portions.

The idea of sub-grammar interaction through simple run-time computed constraints in a conjunctive representation resembles query optimization with projections in database systems (Ullman, 1988).

In the literature there are examples of efficient and compact representations for string sets where strings or tuples contain don't care portions. These include Patricia trees (Morrison, 1968) and *binary decision diagram*s (BDDs) (Bryant, 1986).

The first linear-time minimization algorithm for acyclic finite automata is due to (Godlevskii and Krivoi, 1986) (*c.f.* Revuz (1991, 1992)) An algorithm that combines the "folding" reduction with an acyclic minimization algorithm has been presented in (Yli-Jyrä, 1997).

A Remark: Implementation The presented approach has been implemented partially (Yli-Jyrä, 1997), but its effectiveness was limited because we did not separate different bracketing levels into different sub-grammars (Yli-Jyrä and Koskenniemi, 2004), which is a more important optimization. When we implemented an experimental parser **SkeemaParser** in 1995–1996 we used only folding reductions, but the advantages of the reductions were lost when we combined reduced SRAs with each other. This is in analogy with the problem of sub-grammars: after they are combined, the state blow-up may occur again. The recent optimization based on subgrammars

makes individual constraints and versions of sentence automata substantially simpler, which makes both vertical (Yli-Jyrä, 2004d) and horizontal (Yli-Jyrä, 1997) simplification methods effective.

(IV) Horizontal Expansion of the Alphabet

Perhaps the most interesting and obvious contribution of the article is the alphabet expansion that is done for the sentence automaton and the SRAs.

Remarks Horizontal alphabet extension (based on string positions) is a very powerful method. In fact, it squeezes more power out of finite techniques than one might have thought: alphabet expansion that is *polynomially* dependent on the size of the input (polynomial to the number of string positions, $n^{O(1)}$) could be used to transform *polynomial time* (PTIME) complete problem *instances* into a form where finite-state techniques could be used to carry out the problem solving. In the world of first order definability and descriptive complexity (figure 3.1) polynomial alphabet extension is closely related to precomputed variables and to the *first order queries with BIT and polynomial number of extended variables* (FO[$n^{O(1)}$]) complexity class (Immerman, 1999). Note, however, that in the article I made only linear alphabet expansion, which is a special case of polynomial expansions.

Another interesting, but not a completely new observation (*c.f.* Medvedev, 1964) concerns the structural complexity of regular languages and homomorphic representations: horizontal and vertical alphabet extensions can be used to represent regular languages with simpler regular languages and a homomorphic mapping.

There is one more reason why extended alphabets are nice. A larger space of symbols in the sentence automata leads to efficient heuristics that can be used to simplify the versions of the sentence automata by computing the largest common subset of their input alphabets and removing all transitions whose label are outside of this subset.

It would be naïve to forget that input alphabet size is also an important source of complexity. It may affect the practicality of finite-state automata because the involved input alphabets can become very large when horizontal alphabet extension is used. We have, however, learned in experience that the size of the input alphabet is a much smaller practical problem than the size of the state set, due to many compression techniques available at the implementation level.

The technique that uses an extended alphabet to encode contexts of each letter is analogous to

- the use of intermediate alphabets in transducer pairs (Roche, 1995),
- cascade decompositions using covers (Zeiger, 1968),
- factored extraction of constraints (Maxwell and Kaplan, 1993), and
- named disjunctions and contexted unification (Maxwell and Kaplan, 1991).

The above techniques provide an extremely flexible framework for computing intersections, but it has been long an open problem how it could be used in the most efficient way. The cure will be the same in both cases: use projections to extract tapes and

positions where sub-grammars overlap and solve the intersection problem separately in this new auxiliary grammar. Afterwards consistency between the sub-grammars will have to be enforced. In *constraint satisfaction problem* (CSP) solving, this approach corresponds to the so-called *cycle cut-set* or *tree-clustering* schemes.

5.5 Other Parsing Strategies

There remain a number of parsing strategies that we do not investigate in this dissertation. These are listed in this section because it is important to know that I have been aware of them when choosing those that are in my articles. Unfortunately, the space does not allow discussing pros and cons of each method.

5.5.1 Searching Methods

Backtracking Search Tapanainen (1997) has presented several methods that search individual strings from the intersection of the sentence automaton and the constraints. His depth-first algorithm that enumerates paths in the direct product of the automata is perhaps better characterized as a backtracking search algorithm.

Depth-first search with thousands of automata and rather long search paths calls for a lot of stack storage, but if the constraint automata were replaced with SRAs, one could perhaps implement a more sparce data structure for the stack as well.

Improved Backtracking Search A backtracking search can be improved using so-called intelligent backtracking (see Bruynooghe, 2004) that caches information about failure causes (memoization). Tapanainen (1997) improves his depth-first search algorithm by combining the sentence automata with a set of carefully selected rules before the depth-first search starts. In fact, I took a further step towards that direction in my master's thesis (Yli-Jyrä, 1997) and in 1997-1998 in my second FSIG parser, by suggesting that automata representing the constraints should be transformed into acyclic automata (by combining the sentence automaton to them) before the search starts. Variable ordering is another optimization technique, which might be implemented with compact *specialized rule automata* (SRAs) (Yli-Jyrä, 1995): the original string position of each letter (a position corresponds to an attribute or variable) is indicated in each extended symbols that is used in SRAs.

Some FSIG parsers allowed soft constraints (Voutilainen, 1994a) that were used to reduce the ambiguity on the basis of heuristic constraints. Soft constraints can be evaluated by using the best-first search strategy, but they seem to be feasible only after the hard constraints have been applied and there remains a reasonably small number of potential parses.

Dynamic Elimination of Useless Constraints Typically there are many constraints that may affect the parsing result only if certain special annotation features are presented in the sentence automaton. Based on this observation, Tapanainen (1993) devised a method that determines dynamically, on a sentence-by-sentence basis, which constraint automata are active *i.e.* can reject a string that is accepted by the original

sentence automaton. Inactive constraint automata can be put aside when the search for parses is performed. Furthermore, more inactive rules can be detected by computing a tentative intersection (Tapanainen, 1997) or an SRA (Yli-Jyrä, 1997).

5.5.2 Automata Construction Methods

Parsing by Breadth-First Intersection Tapanainen (1993) has investigated some intersection algorithms that computed the initial-state accessible states in the direct product of the sentence automaton and all constraint automata. Some of his algorithms worked in the breadth-first order. He also investigated the construction of the minimized automaton through union and complementation (Tapanainen, 1991).

The transition matrix of the huge automata can be represented compactly with BDDs. Symbolic BDD-based breadth-first traversal methods are discussed in (Cabodi and Camurati, 1997). Representing FSIG very compactly as a LF-AFA (Salomaa and Yu, 2000) was suggested by Yli-Jyrä (2003a). Intersection of a LF-AFAs with the sentence automaton resembles a breadth-first search.

Optimizing the Order of Constraint Application The direct product of the sentence automaton and the constraint automata, or its minimized equivalent, can be computed in a pairwise manner, by combining two automata at a time. In the parser, this can be done by combining a constraint automaton with the current version of the sentence automaton, which gives rise to a sequence of restricted versions of the sentence automaton. The order in which the constraints are applied to the sentence automaton can make considerable differences to the parser's efficiency. Various strategies for selecting an economical order on the sentence-by-sentence basis have already been investigated (Tapanainen, 1991, 1993, 1997).

Finite Cover Automata In FSIG parsing, the length of deep strings are linearly bounded by the length of the input sentence. Thus, the set of potential deep strings is finite. Minimal *deterministic finite cover automata* (DCFAs) are an optimized representation for finite languages (Câmpeanu et al., 1999) and the set of DFCAs is effectively closed under language intersection (Câmpeanu et al., 1999), which suggests a possible application in FSIG parsing.

Optimizations in the Grammar Optimizations in the grammar preprocessing can lead to substantial improvements as we will demonstrate in this chapter. Tapanainen started investigation of grammar optimizations by considering techniques that helped to find constraint automata whose combination remain small (Tapanainen, 1992). These combination techniques are complemented with techniques that decompose constraints into a conjunction of simpler constraints. My first compilation algorithm for context restrictions (Yli-Jyrä, 2003a) could be used for that purpose.

Furthermore, there are techniques that simplify constraint automata under state compatibility that is determined by computing tentatively a direct product of two constraint automata. The tentatively computed product can interpreted as an incompletely

specified automaton and minimized using appropriate algorithms. The approach resembles simplification through folding (Yli-Jyrä, 1995), with three exceptions: (i) It can merge states that do not share common paths, (ii) it uses state covers rather than state partitions, and (iii) it does not take advantage of alphabet expansions. Tapanainen (1992) discusses simplifications that just erase states of an automaton on the basis of a tentative intersection with some fixed constraint (*c.f.* our reference automaton).

Local Reductions In 1995-1996, I implemented a parser, called the **SkeemaParser**, which contained the reduction method ("folding") for SRA (Yli-Jyrä, 1995, 1997). It turned out that although the method often reduced the size of SRA to a tenth of the original, the benefits did not carry over to the intersection of several SRAs. Because almost all constraints interacted with bracketing, reductions of automata failed to decompose the whole intersection problem into small sub-problems that would have been persistent.

5.5.3 Parsing with Consistency Enforcing Techniques

Basic Definitions Automata are a useful representation for constraint relations (Vempaty, 1992; Amilhastre et al., 2001; Yli-Jyrä, 2001), and they can be used for solving CSPs (Vempaty, 1992). It should be noted, however, that CSP solving through automata is a more general problem than FSIG parsing, because each FSIG is fixed rather than a part of the parser's input.

We can also reduce FSIG parsing to CSPs. First we transform the constraints into deterministic acyclic automata as explained in section 5.5.1, then add padding to make the recognized strings equal in length. After this, we interpret each *position* in padded deep strings as a constrained *attribute*. We can now either (i) interpret the *sets* recognized by the acyclic automata as constraint relations over these attributes, (ii) align the states in each automaton, let attributes represent the source and target states and input letters at each position, and represent transitions through *ternary relations* over these attributes.

Efficient CSP solvers first apply so-called consistency enforcing techniques in order to simplify the constraint relations and domains of attribute values. After this, if necessary, they solve the problem with a complete and sound brute-force method such as a backtracking search or synthesis of combined constraints. The simplest consistency enforcing techniques include algorithms for *node consistency* (NC), *arc consistency* (AC) and *path consistency* (PC). The widely known *constraint propagation* (CP) algorithm is a combination of NC and AC techniques.

An Alternating Method Tapanainen (1997) implements a heuristic method that alternates between a synthesis method and a compression method: During the synthesis phase, several constraints are applied to the sentence automaton. When the result grows too big, a compression algorithm compresses the sentence automaton by merging alternative states.

Approaches Based on Strict Local Testability This approach (ii) has been applied to the KIMMO model in two-level morphology (Barton, Jr., 1986b; Sproat, 1992) (Barton, Jr., 1986a; Barton, 1987; Sproat, 1992). Their approach applied node consistency over the attributes corresponding to states and letters (moreover, they simultaneously kept relations representing automata transitions consistent). A similar method could be implemented without explicit attributes if we expand the alphabet of the acyclic automata in the manner proposed in (Yli-Jyrä, 2001, 2003b)[8]. Piitulainen (1995b) implements a more general heuristics that would enforce consistency beyond adjacent positions to an extended neighborhood. This generalization is still not general enough for FSIG grammars because it is at its best when the FSIG constraint languages are *strictly locally testable* (McNaughton and Papert, 1971; de Luca and Restivo, 1980; Caron, 2000)[9].

A Bounded Tree-width Approach Yli-Jyrä (2001) starts from the approach (i) (where padded finite languages are constraint relations), but sketches a method that would decompose the constraint relations, represented by automata, into smaller relations. The assumption was that a decomposition into binary relations would be possible[10], and furthermore, that the resulting constraint graph (a graph showing the dependencies between the positions) would have bounded tree-width. However, these assumptions are unrealistic: even dependency grammars, which are otherwise very simple, would require more expressive data structures[11].

An Attempt to Capture Elementary Trees When I had started my investigations in 2002 (Yli-Jyrä, 2003b), I made a conjecture that in the reduction to a CSP it would be helpful to decompose constraint relations in a more general fashion compared to Yli-Jyrä (2001). The conjecture was motivated by an analogy to context-free parsing using a CP technique (Morawietz, 2000a,b; Blanche and Morawietz, 2000). At that time, I was aware of the fact that CSP with a bounded tree width is solvable in linear time according to the number of nodes in the tree, which means that I realized that I was not trying to replace FSIG parsing with an intractable CSP instance.

Unfortunately the analogy between a CSP over items in parse forests and a CSP over string positions turned out to be imprecise: in Morawietz' approach, bounded sized items corresponded to subtrees covering an unbounded number of strings, while bounded string subsequences would correspond to a set of very small trees or to discontinuous patterns that are not trees at all. What was missing in constraints over

[8] Based on our experiments on an equivalent method, half of the word readings could be removed in this way.

[9] The attribute 'strictly' is essential: *locally testable languages* are Boolean combinations of strictly locally testable ones, and *strongly locally testable* languages are Boolean combinations of a subset of locally testable ones. It is probable that Piitulainen's approach would not work very well if constraint languages belonged to these more general classes but fail to be strictly locally testable.

[10] This is true only for a very restricted subclass of FSIGs, where the constraints are definable by existentially quantified two-variable formulas.

[11] The balanced bracketing used in our representation is not expressible with attribute pairs. However, if the length of sentences was bounded, we could index the word positions in the sentence with a finite set of symbols. This would make government relation expressible by a cubic number of binary relations over pairs of attributes.

string positions was the ability say: “whatever there is between these two positions, if it contains any brackets, that bracketing must be balanced.”

5.6 Conclusions

Chapter 5 has discussed optimizations in FSIG parsing on the basis of the accompanying articles and three problems that are related to compact representation of ambiguity.

The first accompanying article presented several compilation methods for regular restriction expressions. For FSIG parsing, a particularly interesting method for obtaining a sub-grammar from the FSIG regular expressions was presented. With this method, a certain exponential blow-up scenario for constraint automata was avoided.

The remaining two articles sketched various simplification techniques that were based on alphabet expansions, reference automata, string morphisms and local automata transformations under (implicit) invariants. The simplification techniques provide an efficient solution to intersection of regular languages without construction of their full direct product. These techniques helped to avoid blow-up of intermediate results in a couple of worst-case scenarios of FSIG parsing.

Alphabet expansions can be used to make the processed languages in FSIG parser structurally simpler. They differentiate the regular languages from the star-free ones (Medvedev, 1964), decrease dot-depth of start-free languages (vertical alphabet extension) and even make the languages in the parser local or piecewise testable (horizontal alphabet extension). These effects of alphabet expansions seems to be a cornerstone for FSIG parsing in overall and they are related to homomorphic representations of regular languages.

Chapter 6

Concluding Remarks and Further Issues

6.1 Interconnections between the Articles

The organization of this dissertation — the fact that it consists of separate articles — implies that every article has its own more specified aim. At the same time articles contribute something to the general goals of this dissertation. The resulting interconnections between the articles are complex.

Some most obvious interconnections have been presented in figure 6.1. In the figure, the topmost three boxes stand for the three problems that have driven the research. The articles are numbered as [1]...[9] like in chapter 1. The three different box styles correspond to the three different problems that are discussed in chapters 3 - 5. The arcs show either how an idea pops up from an article or how an idea leads to further ideas and articles. The number of connections and ideas available are reduced in this pictorial presentation for simplicity.

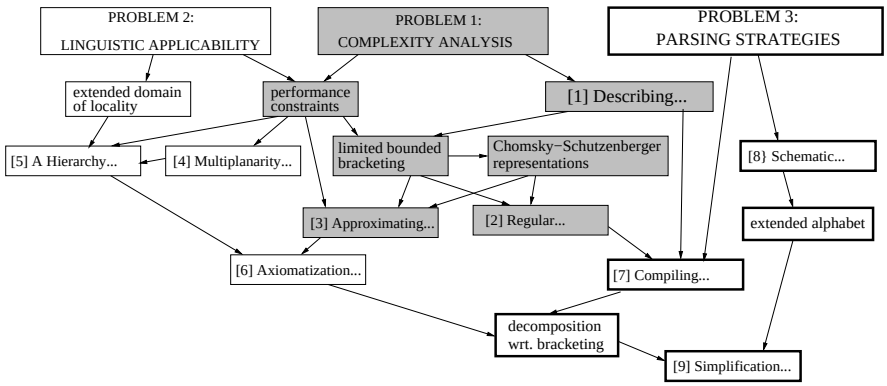


Figure 6.1: A (slightly more) detailed overview of the dissertation.

6.2 Gained Insight

The most important findings in this dissertation can be summarized as follows:

- The constraint used in the new classes of *star-free finite-state intersection grammars* FSIG correspond to languages that do not make use of the Kleene star or second-order quantification at all. *Limited bounded bracketing* (LBB) in deep strings provides a backbone that allows encoding of various properties of sentences using constraints on the bracketing. In addition to phrase-structure bracketing, there is a special bracketing for word-to-word *dependency links*.
- There are relevant *correspondences* between depth of bracketing, dot-depth, descriptive and parallel computational complexity, exponential state-complexity, and succinct representations for constraints and parse forests. Computational complexity of approximations of context-free grammars and mildly context-sensitive grammars remarkably *coincide* in the dot-depth hierarchy.
- New *complexity measures* for dependency trees have been proposed and sufficient conditions for their *acyclicity* has been presented. Structures generated by context-free phrase-structure and dependency grammars and non-projective dependency grammars can be *approximated* with FSIG in a very accurate manner, and parsed in linear time.
- New compilation methods based on *homomorphisms of regular languages* instead of general transducers provide new approaches for compilation of many regular expression operations in natural language processing. *Alphabet expansion* (or extension) and its cancellation with *homomorphisms* is also a cornerstone for compact and flexible representation of intermediate results in parsing.

6.3 Some Future Directions

This dissertation prompts many further theoretical and experimental follow-up research areas. We can only mention here a few possibilities.

In mathematical linguistics, our findings related to star-freeness and expanded alphabets can lead to finer results on the low complexity of representations of natural language grammars and parsers. Parameterized complexity of syntactic structures in natural language and alternative hierarchies of *e.g.* mildly context-sensitive grammars gives rise to exciting opportunities: Psycholinguistic and corpus-based investigations of structural complexity of natural languages could be based on such hierarchies. The obtained complexity restrictions could be of great value to unsupervised procedures for learning FSIG grammars from texts. Because bracketed representations provide regular FSIG approximations in a straightforward manner, development of Chomsky-Schützenberger style homomorphic representations for all important mildly context sensitive grammars could be a well-defined and motivated objective for new researchers. Automatic extraction of approximated mildly context-sensitive grammars from treebanks could accelerate development of large-scale FSIG grammars considerably or accelerate processing of treebank queries (*c.f.* Yli-Jyrä, 2005c). Furthermore,

new bracket-based representations might also be developed for one-level computational morphology and phonology (*c.f.* Yli-Jyrä and Niemi, 2005). For ordinary linguists, development of a flexible and high-level FSIG rule formalism would increase accessibility and attractiveness of the FSIG framework. Such a formalism could incorporate various descriptive approaches and encapsulate axiomatisations and bracket-based representations in an elegant and uniform manner.

Despite of these abundant possibilities for further research, it is even more important that the grammars and parsing strategies presented in this dissertation become implemented and widely accessible. An efficient and usable FSIG parser would guarantee a growing interest in the framework and allow practical evaluation of the methods that has now been presented. Development of an FSIG parser would demand, however, resources that we do not currently have.

To conclude, this dissertation would be most satisfactory if it would inspire focused research and development projects on FSIG approach, facilitate application of related funding and trigger development of linguistically and practically appropriate, widely usable finite-state based environment for a large group of computational linguists.

Bibliography

Abraham, S. 1965. Some questions of language theory. In *COLING 1965, Proceedings of the Conference*, pages 1–11. Bonn, Germany.

Amilhastre, Jérôme, Philippe Janssen, and Marie-Catherine Vilarem. 2001. FA minimisation heuristics for a class of finite languages. In O. Boldt and H. Jürgensen, eds., *WIA'99*, no. 2214 in LNCS, pages 1–12. Berlin and Heidelberg: Springer-Verlag.

Anselmo, Marcella, Dora Giammarresi, and Stefano Varricchio. 2003. Finite automata and non-self-embedding grammars. In J.-M. Champarnaud and D. Maurel, eds., *CIAA 200*, no. 2608 in LNCS, pages 47–56. Berlin and Heidelberg: Springer-Verlag.

Arnola (Jäppinen), Harri. 1998. On parsing binary dependency structures deterministically in linear time. In S. Kahane and A. Polguère, eds., *COLING-ACL'98 Workshop on Processing of Dependency-Based Grammars, Proceedings of the Workshop*, pages 68–77. Montreal.

Bar-Hillel, Yehoshua, Micha Perles, and Eliyahu Shamir. 1964. On formal properties of simple phrase structure grammars. In Y. Bar-Hillel, ed., *Language and Information: Selected Essays on their theory and application*, Addison-Wesley Series in Logic, chap. 9, pages 116–150. Reading, Massachusetts: Addison-Wesley Publishing and Jerusalem, Israel: The Jerusalem Academic Press.

Bar-Hillel, Yehoshua and Eliyahu Shamir. 1964. Finite-state languages: Formal representations and adequacy problems. In Y. Bar-Hillel, ed., *Language and Information. Selected Essays on their Theory and Application*, Addison-Wesley Series in Logic, chap. 7, pages 87–98. Reading, Massachusetts: Addison-Wesley Publishing and Jerusalem, Israel: The Jerusalem Academic Press.

Barton, G. Edward. 1987. The complexity of two-level morphology. In G. E. Barton, R. Berwick, and E. S. Ristad, eds., *Computational Complexity and Natural Languages*, chap. 5, pages 115–186. Cambridge, Massachusetts: A Bradford Book, The MIT Press.

Barton, Jr., G. Edward. 1986a. Computational complexity in two-level morphology. In *24th ACL 1986, Proceedings of the Conference*, pages 53–59. New York, NY, USA: Columbia University.

Barton, Jr., G. Edward. 1986b. Constraint propagation in Kimmo systems. In *24th ACL 1986, Proceedings of the Conference*, pages 42–52. New York, NY, USA: Columbia University.

Becker, Tilman, Owen Rambow, and Michael Niv. 1992. The derivational generative power of formal systems or scrambling is beyond LCFRS. IRCS Report 92-38, Institute for Research in Cognitive Science, University of Pennsylvania, Philadelphia.

Birget, Jean-Camille. 1991a. Errata to "Intersection of regular languages and state complexity". *SIGACT News* 22(3):51.

Birget, Jean-Camille. 1991b. Intersection of regular languages and state complexity. *SIGACT News* 22(2):49.

Birget, Jean-Camille. 1992. Intersection and union of regular languages and state complexity. *Information Processing Letters* 43(4):185–190.

Blanche, Philippe and Frank Morawietz. 2000. A non-generative constraint-based formalism.

Bohnet, Bernd. 2003. Mapping phrase structures to dependency structures in the case of (partially) free word order languages. In *Meaning-Text Theory (MTT) 2003*, pages 217–226. Ecole Normale, Paris, France.

Bowman, Howard and Simon Thompson. 2000. A complete axiomatization of interval temporal logic with projection. Technical Report 6-00, Computing Laboratory, University of Kent.

Bruynooghe, Maurice. 2004. Enhancing a search algorithm to perform intelligent backtracking. *Theory and Practice of Logic Programming (TLP)* 4(3):371–380.

Bryant, Randal E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers (IRE Transactions on Electronic Computers)* C-35(8):677–691. (A retyped version is available through CiteSeer).

Brzozowski, Janusz A. and Robert Knast. 1978. The dot-depth hierarchy of star-free languages is infinite. *Journal of Computer and System Sciences* 16:37–55.

Büchi, J. R. 1960. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 6:66–92.

Cabodi, Gianpiero and Paolo Camurati. 1997. Symbolic FSM traversals based on the transition relation. *IEEE Transactions on Computers (IRE Transactions on Electronic Computers)* 16(5):448–457.

Câmpeanu, Cezar, Nicolae Sântean, and Sheng Yu. 1999. Minimal cover-automata for finite languages. In J.-M. Champarnaud, D. Maurel, and D. Ziadi, eds., *Third International Workshop on Implementing Automata (WIA'98)*, vol. 1660 of *LNCS*, pages 32–42. Berlin and Heidelberg and New York: Springer-Verlag.

Caron, Pascal. 2000. Families of locally testable languages. *Theoretical Computer Science* 242:361–376.

Chanod, Jean-Pierre and Pasi Tapanainen. 1994. Statistical and constraint-based taggers for French. Tech. Rep. MLTT-016, Rank Xerox Research Centre, Grenoble Laboratory, Grenoble, France.

Chanod, Jean-Pierre and Pasi Tapanainen. 1995a. A lexical interface for finite-state syntax. Mltt technical report, Rank Xerox Research Centre, Grenoble Laboratory, Grenoble, France.

Chanod, Jean-Pierre and Pasi Tapanainen. 1995b. Tagging French - comparing a statistical and a constraint-based method. In *7th EACL 1995, Proceedings of the Conference*, pages 149–156. Dublin, Ireland.

Chanod, Jean-Pierre and Pasi Tapanainen. 1996a. A robust finite-state parser for French. In *Proceedings of the ESSLLI'96 Robust Parsing Workshop*, pages 16–25. Prague, Czech.

Chanod, Jean-Pierre and Pasi Tapanainen. 1996b. Rules and constraints in a French finite-state grammar. Mltt technical report, Rank Xerox Research Centre, Grenoble Laboratory, Grenoble, France.

Chanod, Jean-Pierre and Pasi Tapanainen. 1999. Finite-state based reductionistic parsing for French. In A. Kornai, ed., *Extended Finite State Models of Language, Proceedings of the*

ECAI'96 Workshop, Studies in Natural Language Processing, pages 72–85. Cambridge University Press.

Chomsky, Noam. 1956. Three models for description of language. *IEEE (IRE) Transactions on Information Theory* IT-2:113–124. Reprinted in *Readings in Mathematical Psychology*, volume II, pages 105–124, New York: John Wiley and Sons, 1965.

Chomsky, Noam. 1957. *Syntactic Structures*. Den Haag and Paris: Mouton.

Chomsky, Noam. 1959a. A note on phrase structure grammars. *Information and Computation (Information and Control)* 2:393–395.

Chomsky, Noam. 1959b. On certain formal properties of grammars. *Information and Computation (Information and Control)* 2(2):137–167.

Chomsky, Noam. 1963. Formal properties of grammars. In R. Luce, R. Bush, and E. Galanter, eds., *Handbook of Mathematical Psychology*, vol. II, pages 323–418. New York: John Wiley and Sons.

Chomsky, Noam and Marcel-Paul Schützenberger. 1963. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, eds., *Computer Programming and Formal Systems*, pages 118–161. Amsterdam: North Holland Publishing Co.

Church, Kenneth. 1980. On parsing strategies and closure. In *18th ACL 1980, Proceedings of the Conference*, pages 107–111. Philadelphia, Pennsylvania, USA.

Church, Kenneth and Ronald Kaplan. 1981. Removing recursion from natural language processors based on phrase structure grammars. In *Modeling Human Parsing Strategies*. University of Texas at Austin.

Church, Kenneth and Ramesh Patil. 1982. Coping with syntactic ambiguity or how to put the block in the box on the table. *Computational Linguistics (American Journal of Computational Linguistics)* 8(3–4):139–149.

Cohen, Rina S. and Janusz A. Brzozowski. 1971. Dot-depth of star-free events. *Journal of Computer and System Sciences* 5:1–16.

Covington, Michael. 1994a. GB theory as Dependency Grammar. Research Report AI-1992-03, Artificial Intelligence Programs, The University of Georgia, Athens, Georgia 30602, U.S.A.

Covington, Michael A. 1990. Parsing discontinuous constituents in dependency grammar (technical correspondence). *Computational Linguistics (American Journal of Computational Linguistics)* 16(4):234–236.

Covington, Michael A. 1992. A dependency parser for variable-word-order languages. In K. R. Billingsley, H. U. B. III, and E. Derohanes, eds., *Computer assisted modeling on the IBM 3090: Papers from the 1989 IBM Supercomputing Competition*, vol. 2, pages 799–845. Athens, Georgia, USA: Baldwin Press.

Covington, Michael A. 1994b. Discontinuous dependency parsing of free and fixed word order: Work in progress. Research Report AI-1994-02, Artificial Intelligence Programs, The University of Georgia, Athens, Georgia 30602, U.S.A.

Covington, Michael A. 1994c. An empirically motivated reinterpretation of dependency grammar. Research Report AI-1994-01, Artificial Intelligence Programs, The University of Georgia, Athens, Georgia 30602, U.S.A.

Covington, Michael A. 2001. A fundamental algorithm for dependency parsing. In *39th Annual ACM Southeast Conference, ACM-SE, Session 2B: Artificial Intelligence II*. The Georgia

Center, University of Georgia, Athens, GA.

de Luca, Aldo and Antonio Restivo. 1980. A characterization of strictly locally testable languages and its application to subsemigroups of a free semigroup. *Information and Control* 44:300–319.

De Roeck, Anne, Roderick Johnson, Margaret King, Michael Rosner, Geoffrey Sampson, and Nilo Varile. 1982. A myth about centre-embedding. *Lingua* 58:327–340.

Dras, Mark, David Chiang, and William Schuler. 2004. On relations of constituency and dependency grammars. *Research on Language and Computation* 2(2):281–305.

Duchier, Denys. 1999. Axiomatizing dependency parsing using set constraints. In *Sixth Meeting on Mathematics of Language (MOL-6). http://www.ps.uni-sb.de/~duchier/drafts/mol6.ps.gz*, pages 115–126. Orlando, Florida.

Eggan, L. C. 1963. Transition graphs and the star height of regular events. *Michigan Mathematical Journal* 10:385–397.

Ehrich, Hans-Dieter. 1972. A note on state minimization of a special class of incomplete sequential machines. *IEEE Transactions on Computers (IRE Transactions on Electronic Computers)* C-21(5):500–502.

Eilenberg, Samuel. 1974. *Automata, Languages, and Machines*, vol. A of *Pure and Applied Mathematics*. New York: Academic Press.

Ejerhed, Eva and Kenneth Church. 1983. Finite-state parsing. In F. Karlsson, ed., *Papers from the Seventh Scandinavian Conference of Linguistics, volume II*, no. 10 in Publications of the Department of General Linguistics, University of Helsinki, pages 410–432. Helsinki, Finland: Helsingin yliopiston monistuspalvelu.

Elgot, Calvin C. 1961. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society* 98(1):21–51.

Elworthy, David. 2000. A finite state parser with dependency structure output. In *Proceedings of Sixth International Workshop on Parsing Technologies (IWPT 2000*. Trento, Italy: Institute for Scientific and Technological Research.

Fagin, R. 1974. Generalized first-order spectra and polynomial-time recognizable sets. In R. Karp, ed., *Complexity of Computation, SIAM-AMS Proc. 7*.

Frick, Markus and Martin Grohe. 2004. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic* 130:3–31.

Gaifman, Haim. 1965. Dependency systems and phrase-structure systems. *Information and Computation (Information and Control)* 8(3):304–337.

Garey, Michael R. and David S. Johnson. 1979. *Computers and Intractability*. Series of Books in the Mathematical Sciences. New York: W. H. Freeman and Company.

Gaubert, Christophe. 2004. Two-dimensional proof-structures and the exchange rule. *Mathematical Structures in Computer Science (MSCS)* 14(1):73–96.

Gibson, Edward. 1998. Linguistic complexity: locality of syntactic dependencies. *Cognition* 68:1–76.

Ginsburg, Seymour and Michael A. Harrison. 1967. Bracketed context-free languages. *Journal of Computer and System Sciences* 1(1):1–23.

Glaßer, Christian. 2001. *Forbidden-Pattern and Word Extensions for Concatenation Hierarchies*. Ph.D. thesis, Bayerischen Julius-Maximilians-Universität, Würzburg.

Godlevskii, A. B. and S. L. Krivoi. 1986. Transformation synthesis of efficient algorithms with auxiliary specifications (English translation from Russian, 1987, Plenum Publishing Corporation). *Kibernetica* 66(6):37–43.

Grasselli, Antonio and Fabrizio Luccio. 1965. A method for minimizing the number of internal states in incompletely specified sequential networks. *IEEE Transactions on Computers (IRE Transactions on Electronic Computers)* EC-14(3):350–359.

Gross, Maurice. 1997. The construction of local grammars. In E. Roche and Y. Schabes, eds., *Finite-State Language Processing*, chap. 11, pages 329–354. Cambridge, MA, USA: A Bradford Book, the MIT Press.

Hale, John and Paul Smolensky. 2001. A parser for harmonic context-free grammars. In *39th ACL 2001, Proceedings of the Conference*, pages 427–432. Toulouse, France.

Harrison, Michael A. 1978. *Introduction to Formal Language Theory*. Addison-Wesley: Reading, Massachusetts.

Hays, David E. 1964. Dependency theory: a formalism and some observations. *Language* 40:511–525.

Heinonen, Tarja Riitta. 1993. Fin-PCG: Suomen Paraller Constraint-kielioppia. Memorandum. Lingsoft Inc.

Höfler, Stefan. 2002. Link2tree: A dependency-constituency converter. Lizentiatsarbeit, Institute of Computational Linguistics, University of Zürich, Zürich.

Holan, Tomáš. 2002. Dependency analyser configurable by measures. In P. Sojka, I. Kopeček, and K. Pala, eds., *TSD 2002*, vol. 2448 of *LNAI*, pages 81–88. Berlin and Heidelberg: Springer-Verlag.

Holan, Tomáš, Vladislav Kubon, Karel Oliva, and Martin Plátek. 1998. Two useful measures of word order complexity. In S. Kahane and A. Polguère, eds., *COLING-ACL'98 Workshop on Processing of Dependency-Based Grammars, Proceedings of the Workshop*, pages 21–28. Montreal.

Hopcroft, John E. and Jeffrey D. Ullman. 1969. *Formal languages and their relation to automata*. Addison-Wesley Series in Computer Science and Information Processing. Reading, M.A.: Addison-Wesley.

Huffman, David A. 1954. The synthesis of sequential switching circuits. *Journal of the Franklin Institute* 257(3–4):161–191, 275–303.

Ihm, P. and Y. Lecerf. 1963. Éléments pour une grammaire générale des langues projectives. (Appeared earlier as Rapport GRISA, number 1, pages 11–29, 1960.) Rapport EUR 210.f, Bruxelles: EURATOM, Communauté Européenne de l'énergie atomique. (Rapport CETIS, Centre de Traitement de l'Information Scientifique), pages 1–24.

Immerman, Neil. 1999. *Descriptive Complexity*. Graduate Texts in Computer Science. New York: Springer-Verlag.

Johnson, Mark. 1996. Left corner transforms and finite state approximation. DRAFT of 12th May, 1996. Rank Xerox Research Centre.

Johnson, Mark. 1998. Finite-state approximation of constraint-based grammars using left-corner grammar transforms. In *36th ACL 1998, 17th COLING 1998, Proceedings of the Conference*, vol. 1, pages 619–623. Montréal, Quebec, Canada.

Joshi, Aravind. 2004. Starting with complex primitives pays off: Complicate locally, simplify

globally. Preprint submitted to Elsevier Science.

Joshi, Aravind K. 1985. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In D. R. Dowty, L. Karttunen, and A. M. Zwicky, eds., *Natural language parsing: psychological, computational and theoretical perspectives*, Studies in Natural Language Processing, chap. 6, pages 206–250. Cambridge University Press.

Joshi, Aravind K., Tilman Becker, and Owen Rambow. 2000. Complexity of scrambling: a new twist to the competence/performance distinction. In A. Abeille and O. Rambow, eds., *Tree Adjoining Grammars: formalisms, linguistic analysis and processing*, pages 167–182. Stanford: CSLI publications.

Joshi, Aravind K. and Leon S. Levy. 1982. Phrase structure trees bear more fruit than you would have thought. *Computational Linguistics (American Journal of Computational Linguistics)* 8(1):1–11.

Jurvanen, Eija, Andreas Potthoff, and Wolfgang Thomas. 1993. Tree languages recognizable by regular frontier check. In G. Rozenberg and A. Salomaa, eds., *Developments in Language Theory, At Crossroads of Mathematics, Computer Science and Biology, Turku, Finland, 12-15 July 1993*, pages 3–17. Singapore: World Scientific. ISBN 981-02-1645-9.

Kallmeyer, Laura and SinWon Yoon. 2004. Tree-local MCTAG with shared nodes: word-order variation in German and Korean. In *Proceedings of TAG+7, Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms*. Vancouver, British Columbia, Canada: Simon Fraser University.

Kaplan, Ronald M. and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20(3):331–378.

Karlsson, Fred. 1990. Constraint grammar as a framework for parsing running text. In H. Karlgren, ed., *13th COLING 1990, Proceedings of the Conference*, vol. 3, pages 168–173. Helsinki, Finland.

Karlsson, Fred. 2004 (in print). Limits of clausal embedding complexity in Standard Average European. Manuscript. Department of General Linguistics, University of Helsinki. April.

Kempe, Andre and Jean-Marc Champarnaud. 2004. Some problems of multi-tape intersection. In L. Cleophas and B. W. Watson, eds., *The Eindhoven FASTAR Days, Proceedings*, no. 04/40 in Computer Science Reports. Eindhoven, The Netherlands: Technische Universiteit Eindhoven.

Kempe, Andre, Frank Guingne, and Florent Nicart. 2004a. Algorithms for weighted multi-tape automata. XRCE Research Report 2004/031, Xerox Research Centre Europe.

Kempe, Andre, Franck Guingne, and Florent Nicart. 2004b. New algorithms for weighted multi-tape automata. In M. Droste and H. Vogler, eds., *Weighted Automata: Theory and Applications, Dresden, Germany, June 1–5, 2004*, no. TUD-FI04-05 in Technische Berichte der Fakultät Informatik, ISSN-1430-211X, pages 42–43. D-01062 Dresden, Germany: Techniche Universität Dresden.

Kiraz, G. 1996a. ṢEMḤE: a generalized two-level system. In *34th ACL 1996, Proceedings of the Conference*, pages 159–166. Santa Cruz, CA, USA.

Kiraz, G. 1997. Compiling regular formalisms with rule features into finite-state automata. In *35th ACL 1997, 8th EACL 1997, Proceedings of the Conference*, pages 329–336. Madrid, Spain.

Kiraz, George Anton. 1994. Multi-tape two-level morphology: A case study in semitic non-

linear morphology. In *15th COLING 1994, Proceedings of the Conference*, vol. 1, pages 180–186. Kyoto, Japan.

Kiraz, George Anton. 1996b. Computing prosodic morphology. In *16th COLING 1996, Proceedings of the Conference*, pages 664–669. Copenhagen, Denmark.

Kiraz, George Anton. 2000. Multitiered nonlinear morphology using multitape finite automata: A case study on Syriac and Arabic. *Computational Linguistics* 26(1):77–105.

Kiraz, George Anton and Edmund Grimley-Evans. 1998. Multi-tape automata for speech and language systems: A prolog implementation. In D. Wood and S. Yu, eds., *Automata Implementation*, no. 1436 in Lecture Notes in Computer Science, pages 87–103. Springer Verlag.

Klarlund, Nils. 1998. Mona & Fido: The logic-automaton connection in practice. In M. Nielsen and W. Thomas, eds., *Computer Science Logic, 11th International Workshop, CSL'97, Annual Conference of the EACSL, Aarhus, Denmark, August 23-29, 1997, Selected Papers*, vol. 1414 of *LNCS*, pages 311–326. Springer.

Kornai, András. 1985. Natural language and the Chomsky hierarchy. In *2nd EACL 1985, Proceedings of the Conference*, pages 1–7. Geneva, Switzerland.

Kornai, András. 1999. Zipf's law outside the middle range. In *Proceedings of the Sixth Meeting on Mathematics of Language*, pages 347–356. University of Central Florida.

Kornai, András. 2002. How many words are there? *Glottometrics* 4:61–86.

Kornai, András and Zsolt Tuza. 1992. Narrowness, path-width, and their application in natural language processing. *Discrete Applied Mathematics* 36:87–92.

Koskenniemi, Kimmo. 1983. *Two-level morphology: a general computational model for word-form recognition and production*. No. 11 in Publications of the Department of General Linguistics, University of Helsinki. Helsinki: Yliopistopaino.

Koskenniemi, Kimmo. 1990. Finite-state parsing and disambiguation. In H. Karlgren, ed., *13th COLING 1990, Proceedings of the Conference*, vol. 2, pages 229–232. Helsinki, Finland.

Koskenniemi, Kimmo. 1997. Representations and finite-state components in natural language. In E. Roche and Y. Schabes, eds., *Finite-state language processing*, chap. 3, pages 99–116. Cambridge, Massachusetts: A Bradford Book, The MIT Press.

Koskenniemi, Kimmo, Pasi Tapanainen, and Atro Voutilainen. 1992. Compiling and using finite-state syntactic rules. In *14th COLING 1992, Proceedings of the Conference*, vol. 1, pages 156–162. Nantes, France.

Kozen, Dexter. 1977. Lower bounds for natural proof systems. In *Proceedings of 18th Annual Symposium on Foundations of Computer Science*, pages 254–266. IEEE Computer Society, Long Beach, California.

Krauwer, Steven and Louis des Tombe. 1980. The finite state transducer as a theory of language. *Utrect Working Papers in Linguistics, UWPL* 9:1–86.

Krauwer, Steven and Louis des Tombe. 1981. Transducers and grammars as theories of language. *Theoretical Linguistics* 8:173–202.

Kromann, M. T., L. Mikkelsen, and S. K. Lynge. 2003. The Danish Dependency Treebank Website. Department of Computational Linguistics, Copenhagen Business School. `http://www.id.cbs.dk/~mtk/treebank`.

Kruijff, Geertjan. 2002. A formal language hierarchy based on dependency grammars. MSc Thesis proposal. Universität des Saarlandes, Posted 26 April, 2002. Revised 8 October 2002.

Lager, Torbjörn. 1999. Logic for part of speech tagging and shallow parsing. In *The 11th Nordic Conference in Computational Linguistics (NoDaLiDa'98)*. Copenhagen.

Lager, Torbjörn and Joakim Nivre. 2001. Part of speech tagging from a logical point of view. In P. de Groote, G. Morrill, and C. Retoré, eds., *Logical Aspects of Computational Linguistics*, vol. 2099 of *Lecture Notes in Artificial Intelligence*, pages 212–227. Springer-Verlag.

Lecerf, Y. 1960. Analyse automatique (programmes de conflits). In A. Leroy, ed., *Enseignement preparatoire aux techniques de la documentation automatique, EUR/C/867/61f*, pages 179–253. Bruxelles: Bruxelles: EURATOM, Communauté Européenne de l'énergie atomique.

Lecerf, Yves. 1961. Une représentation algébrique de la structure des phrases dans diverses langues naturelles. Notes in Comptes Rendus [hebdomadaires] des Séances de l'Academie des Sciences, Éditeur-Imprimeur-Libraire, 252(2 February):232–234, Paris: Gauthier-Villars & C^{ie}.

Leiss, Ernst. 1991. Some comments on a recent note of Ravikumar. *SIGACT News* 22(1):64.

Lewis, Richard L. 1996. Interference in short-term memory: The magical number two (or three) in sentence processing. *Journal of Psycholinguistic Research* 25(1):93–116.

Lounela, Mikko. 1993. Feature based parsing. In *Datalingvistisk symposium for forskerrekrutter*, pages 47–52. NORFA, Copenhagen.

Marcus, Solomon. 1965. Sur la notion de projectivité. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 11:181–192.

Marcus, Solomon. 1967. *Algebraic Linguistics; Analytical Models*, vol. 29 of *Mathematics in Science and Engineering*, chap. VI (Subordination and Projectivity), pages 200–246. New York and London: Academic Press.

Maruyama, Hiroshi. 1990. Structural disambiguation with contraint propagation. In *28th ACL 1989, Proceedings of the Conference*, pages 31–38. Pittsburgh, Pennsylvania.

Maxwell, John T. III and Ronald M. Kaplan. 1991. A method for disjunctive constraint satisfaction. In M. Tomita, ed., *Current Issues in Parsing Technology*, pages 173–190. Kluwer Academic Publishers.

Maxwell, John T. III and Ronald M. Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics* 19(4):571–590.

Maxwell, John T. III and Ronald M. Kaplan. 1996. An efficient parser for LFG. In *Proceedings of the First LFG Conference*, page 1. Grenoble, France.

McCluskey, Edward J. 1962. Minimum-state sequential circuits for a restricted class of incompletely specified flow tables. *The Bell System Technical Journal* XLI(6):1759–1768.

McNaughton, Robert and Seymour Papert. 1971. *Counter-Free Automata*. No. 65 in Research Monograph. Cambridge, Massachusetts: MIT Press.

Medvedev, Yu. T. 1964. On the class of events representable in a finite automaton (reprint of lincoln laboratories group report 34-73 (1958), an English translation from the Russian version that appeared in 1956). In *Sequential Machines – Selected Papers*, pages 215–227. Addison-Wesley.

Meyer, A. R. 1969. A note on star-free events. *Journal of the ACM* 16(2):220–225.

Meyer, A. R. 1975. Weak monadic second order theory of successor is not elementary-recursive. In R. Parikh, ed., *Logic Colloquium (Proc. Symposium on Logic, Boston, 1972)*, vol. 453 of *LNCS*, pages 132–154. Springer.

Miller, George A. and Noam Chomsky. 1963. Finitary models of language users. In R. D. Luce, R. R. Bush, and E. Galanter, eds., *Handbook of Mathematical Psychology*, vol. II, pages 419–491. John Wiley.

Miller, Philip H. 2000. *Strong Generative Capacity: The Semantics of Linguistic Formalism*. Stanford: CSLI Publications.

Mohri, Mehryar. 1995. Matching patterns of an automaton. In Z. Galil and E. Ukkonen, eds., *Combinatorial Pattern Matching, 6th Annual Symposium, CMP 95, Proceedings*, no. 937 in LNCS, pages 286–297. Espoo, Finland: Springer.

Morawietz, Frank. 2000a. Chart parsing and constraint programming. In *20th COLING 2000, Proceedings of the Conference*, pages 551–557. Saarbrücken, Germany.

Morawietz, Frank. 2000b. Chart parsing as constraint propagation. In *Some Aspects of Natural Language Processing and Constraint Programming*, Arbeitspapiere des Sonderforschungsbereichs 340, Bericht Nr. 150, pages 29–50. Universität Stuttgart and Universität Tübingen and IBM Deutschland.

Morrison, Donald R. 1968. PATRICIA – Practical Algorithm To Retrieve Information Coded in Alphanumeric. *Journal of the ACM* 15(4):514–534.

Nasr, Alexis and Owen Rambow. 2004. A simple string-rewriting formalism for dependency grammar. In G.-J. M. Kruijff and D. Duchier, eds., *Proc. Workshop of Recent Advances in Dependency Grammar*, pages 33–40. Geneva, Switzerland.

Nederhof, Mark-Jan. 2000. Practical experiments with regular approximation of context-free languages. *Computational Linguistics* 26(1):17–44.

Nivre, Joakim. 2003a. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies, IWPT 2003*, pages 149–160. Nancy, France.

Nivre, Joakim. 2003b. Theory supporting treebanks. In J. Nivre and E. Hinrichs, eds., *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT 2003)*, no. 9 in Mathematical Modelling in Physics, Engineering and Cognitive Sciences, pages 117–128. Växjö: Växjö University Press.

Nivre, Joakim and Mario Scholz. 2004. Deterministic dependency parsing of english text. In *20th COLING 2004, Proceedings of the Conference*, vol. I, pages 64–70. Geneva, Switzerland.

Oflazer, Kemal. 2003. Dependency parsing with an extended finite-state approach. *Computational Linguistics* 29(4):515–544.

Pager, David. 1971. Conditions for the existence of minimal closed covers composed of maximal compatibles. *IEEE Transactions on Computers (IRE Transactions on Electronic Computers)* C-20:450–452.

Paull, Marvin C. and Stephen H. Unger. 1959. Minimizing the number of states in incompletely specified sequential switching functions. *IEEE Transactions on Computers (IRE Transactions on Electronic Computers)* EC-8(3):356–367.

Paull, Marvin C. and G. Waldbaum. 1967. A note on state minimization of asynchronous sequential functions. *IEEE Transactions on Computers (IRE Transactions on Electronic Computers)* EC-16(1):94–97.

Perrin, Dominique and Jean-Eric Pin. 2001. Infinite words. Manuscript. (The final version published by Elsevier in 2004).

Peters, Paul Stanley and Robert W. Ritchie. 1969. Context sensitive immediate constituent analysis — context-free languages revisited. In *Proc. ACM Symposium on Theory of Computing*, pages 1–8. Marina del Rey, California.

Pfleeger, Charles P. 1973. State reduction in incompletely specified finite state machines. *IEEE Transactions on Computers (IRE Transactions on Electronic Computers)* C-22(12):1099–1102.

Piitulainen, Jussi. 1995a. *Kielioppijärjestelmien kokonaisvertailu (Comparison of Grammar Systems)*. Master's thesis, Department of General Linguistics, University of Helsinki, Helsinki, Finland.

Piitulainen, Jussi. 1995b. Locally tree-shaped sentence automata and resolution of ambiguity. In *Proceedings of the 10th Nordic Conference of Computational Linguistics (NODALIDA-95)*, no. 26 in Publications of the Department of General Linguistics, University of Helsinki, pages 50–58. Helsinki, Finland: Yliopistopaino.

Pin, Jean-Eric. 1986. *Varieties of Formal Languages*. Foundations of Computer Science. London and New York: North Oxford Academic Publishers and Plenum Press.

Pin, Jean-Eric. 2003. Algebraic tools for the concatenation product. *Theoretical Computer Science* 292(1):317–342.

Popper, Karl Raimund. 1959. *The Logic of Scientific Discovery*. London: Hutchinson & Co., 2nd edn.

Post, Emil. 1943. Formal reductions of the general combinatory decision problem. *American Journal of Mathematics* 65:197–215.

Pullum, Geoffrey and Gerald Gazdar. 1982. Natural languages and context-free languages. *Linguistics and Philosophy* 4(4):471–504.

Pullum, Geoffrey K. 1984. Syntactic and semantic parsability. In *10th COLING 1984, Proceedings of the Conference*, pages 112–122. Stanford, California.

Pullum, Geoffrey K. and Barbara Scholz. 2003. Model-theoretic syntax foundations – linguistic aspects. Lecture notes, 15th European Summer School in Logic, Language, and Information, August 18-19, at Technical University of Vienna.

Rabin, Michael O. and Dana Scott. 1959. Finite automata and their decision problems. *IBM Journal of Research and Development* 3(2):114–125.

Ravikumar, Bala. 1990. Some applications of a technique of Sakoda and Sipser. *SIGACT News* 21(4):73–77.

Revuz, Dominique. 1991. *Dictionaires et lexiques, méthodes et algorithmes*. Ph.D. thesis, Institut Blaise Pascal, Université Paris 7, Paris.

Revuz, Dominique. 1992. Minimization of acyclic deterministic automata in linear time. *Theoretical Computer Science* 92(1):181–189.

Robinson, Jane J. 1967. Methods for obtaining corresponding phrase structure and dependency grammars. In *Proceedings of the 1967 conference on Computational Linguistics*, pages 1–25.

Roche, Emmanuel. 1995. Smaller representations for finite-state transducers and finite-state automata. In Z. Galil and E. Ukkonen, eds., *Combinatorial Pattern Matching, 6th Annual Symposium, CMP 95, Proceedings*, no. 937 in LNCS, pages 352–365. Espoo, Finland: Springer.

Roche, Emmanuel. 1997. Parsing with finite-state transducers. In E. Roche and Y. Schabes, eds., *Finite-state language processing*, chap. 8, pages 241–281. Cambridge, Massachusetts:

A Bradford Book, the MIT Press.

Rogers, James. 1998. *A descriptive approach to language-theoretic complexity*. Studies in Logic, Language and Information. Stanford, United States: CSLI Publications & FoLLI.

Rogers, James. 2003. Syntactic structures as multi-dimensional trees. *Research on Language and Computation* 1:265–305.

Rogers, James. 2004. On scrambling, another perspective. In *Proceedings of TAG+7, Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms*, pages 178–185. Vancouver, British Columbia, Canada: Simon Fraser University.

Salomaa, Kai and Sheng Yu. 2000. Alternating finite automata and star-free languages. *Theoretical Computer Science* 234:167–176.

Schützenberger, Marcel Paul. 1965. On finite monoids having only trivial subgroups. *Information and Computation (Information and Control)* 8(2):190–194.

Schwentick, Thomas and Klaus Barthelmann. 1999. Local normal forms for first-order logic with applications to games and automata. *SIAM Journal on Discrete Mathematics (SIDMA)* 3:109–124.

Simon, Imre. 1975. Piecewise testable events. In *Proceedings of the 2nd GI Conference*, vol. 33 of *Lecture Notes in Computer Science*. Berlin: Springer.

Sleator, Daniel and Davy Temperley. 1991. Parsing English with a link grammar. Technical Report CMU-CS-91-196, Carnegie Mellon University, Computer Science, Pittsburgh, USA.

Sproat, Richard William. 1992. *Morphology and Computation*, pages 171–179. Cambridge, Massachusetts: The MIT Press.

Stearns, R. E. 1967. A regularity test for pushdown machines. *Information and Control* 11(3):323–340.

Tamm, Hellis. 2004. On minimality and size reduction of one-tape and multitape finite automata. Series of Publications A Report A-2004-9, Department of Computer Science, University of Helsinki.

Tamm, Hellis, Matti Nykänen, and Esko Ukkonen. 2004. Size reduction of multitape automata. A Poster Presented at CIAA 2004.

Tapanainen, Pasi. 1991. Äärellisinä automaatteina esitettyjen kielioppisääntöjen soveltaminen luonnollisen kielen jäsentäjässä. Master's thesis C-1992-05, Department of Computer Science, University of Helsinki, Helsinki, Finland.

Tapanainen, Pasi. 1992. Äärellisiin automaatteihin perustuva luonnollisen kielen jäsennin. Licentiate thesis C-1993-07, Department of Computer Science, University of Helsinki, Helsinki, Finland.

Tapanainen, Pasi. 1993. Finite state parsing. In *Datalingvistisk symposium for forskerrekrutter*, pages 1–9. NORFA, Copenhagen.

Tapanainen, Pasi. 1997. Applying a finite-state intersection grammar. In E. Roche and Y. Schabes, eds., *Finite-State Language Processing*, chap. 10, pages 311–327. Cambridge, MA, USA: A Bradford Book, the MIT Press.

Tapanainen, Pasi. 1999. *Parsing in two frameworks: finite-state and functional dependency grammar*. Ph.D. thesis, University of Helsinki, Finland.

Teich, Elke. 1998. Types of syntagmatic relations and their representation. In S. Kahane and A. Polguère, eds., *36th ACL 1998, 17th COLING 1998, Proceedings of the Conference*, vol. 1,

pages 39–48. Montréal, Quebec, Canada.

Teitelman, Warren. 1978. *INTERLISP Reference Manual*. Xerox Palo Alto Research Center, Xerox Corporation. (Section 2.2 Using Interlisp – an Overview, 2.4).

Tesnière, Lucien. 1969 (1959). *Éléments de Syntaxe Structurale*. Paris: Éditions Klincksieck, 2nd edn.

Thatcher, James W. 1967. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences* 1:317–322.

Thomas, Wolfgang. 1982. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences* 25:360–376.

Thomas, Wolfgang. 1997. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages*, pages 389–455. Springer.

Tomescu, Ioan. 1972. A method for minimizing the number of states for a restricted class of incompletely specified sequential machines. *Mathematical Systems Theory* 6(1):1–2.

Ullian, Joseph S. 1967. Partial algorithm problems for context free languages. *Information and Computation (Information and Control)* 11:80–101.

Ullman, Jeffrey D. 1988. *Principles of Database and Knowledge-Base Systems*, vol. 1-2. New York: Computer Science Press.

van Melkebeek, Dieter. 2000. *Randomness and completeness in computational complexity*. ACM Distinguished Theses. Springer-Verlag.

Vempaty, N. R. 1992. Solving constraint satisfaction problems using finite state automata. In *American Association for Artificial Intelligence (AAAI'92)*, pages 453–458. San Jose.

Voutilainen, Atro. 1994a. *Designing a Parsing Grammar*. No. 22 in Publications of the Department of General Linguistics, University of Helsinki. Helsinki, Finland: Yliopistopaino.

Voutilainen, Atro. 1994b. *Three studies of grammar-based surface parsing of unrestricted English text*. No. 24 in Publications of the Department of General Linguistics, University of Helsinki. Helsinki, Finland: Yliopistopaino.

Voutilainen, Atro. 1997. Designing a (finite-state) parsing grammar. In E. Roche and Y. Schabes, eds., *Finite-State Language Processing*, chap. 9, pages 283–310. Cambridge, MA, USA: A Bradford Book, the MIT Press.

Voutilainen, Atro. 1998. Does tagging help parsing? A case study on finite state parsing. In L. Karttunen, ed., *FSMNLP'98: International Workshop on Finite State Methods in Natural Language Processing*, pages 25–36. Somerset, New Jersey.

Voutilainen, Atro and Pasi Tapanainen. 1993. Ambiguity resolution in a reductionistic parser. In *6th EACL 1993, Proceedings of the Conference*, pages 394–403. Utrecht, The Netherlands.

Wartena, Christian. 2001. Grammars with composite storages. In M. Moortgat, ed., *LACL'98*, vol. 2014 of *LNAI*, pages 266–285. Springer.

Weir, David. 1992. A geometric hierarchy beyond context-free languages. *Theoretical Computer Science* 104(4):235–261.

Wintner, Shuly. 2001. Formal language theory for natural language processing. A foundational course taught in European Summer School in Logic, Language and Information, August 13-24, Helsinki. Available at `http://www.helsinki.fi/esslli/courses/readers/K10.pdf`.

Yli-Jyrä, Anssi Mikael. 1995. Schematic finite-state intersection parsing. In K. Koskenniemi, ed., *Short Papers Presented at the 10th Nordic Conference of Computational Linguistics (NODALIDA-95)*, pages 95–103. Helsinki, Finland. **[9]**.

Yli-Jyrä, Anssi Mikael. 1997. *Menetelmiä äärellisiin automaatteihin perustuvan lauseenjäsennyksen tehostamiseksi*. Master's thesis, Department of General Linguistics, University of Helsinki, Helsinki, Finland.

Yli-Jyrä, Anssi Mikael. 2001. Structural correspondence between finite-state intersection grammar and constraint satisfaction problem, extended abstract. In *Finite State Methods in Natural Language Processing 2001 (FSMNLP 2001), ESSLLI Workshop*, pages (1–4). Helsinki.

Yli-Jyrä, Anssi Mikael. 2003a. Describing syntax with star-free regular expressions. In *11th EACL 2003, Proceedings of the Conference*, pages 379–386. Agro Hotel, Budapest, Hungary. **[1]**.

Yli-Jyrä, Anssi Mikael. 2003b. Efficient parsing with finite-state constraint satisfaction, a Ph.D. project. In H. Holmboe, ed., *Nordisk Sprogteknologi 2002. Nordic Language Technology 2002. Årbog for Nordisk Sprogteknologisk Forskningsprogram 2000–2004*, pages 427–430. Copenhagen: Museum Tusculanums Forlag.

Yli-Jyrä, Anssi Mikael. 2003c. Multiplanarity – a model for dependency structures in treebanks. In J. Nivre and E. Hinrichs, eds., *TLT 2003. Proceedings of the Second Workshop on Treebanks and Linguistic Theories*, vol. 9 of *Mathematical Modelling in Physics, Engineering and Cognitive Sciences*, pages 189–200. Växjö, Sweden: Växjö University Press. **[4]**.

Yli-Jyrä, Anssi Mikael. 2003d. Regular approximations through labeled bracketing. In G. Jäger, P. Monachesi, G. Penn, and S. Wintner, eds., *Proceedings of the 8th conference on Formal Grammar 2003 "FG Vienna"*, pages 189–201. Pre-proceedings. Available at `http://cs.haifa.ac.il/~shuly/fg03/`.

Yli-Jyrä, Anssi Mikael. 2004a. Axiomatization of restricted non-projective dependency trees through finite-state constraints that analyse crossing bracketings. In G.-J. M. Kruijff and D. Duchier, eds., *Proc. Workshop of Recent Advances in Dependency Grammar*, pages 33–40. Geneva, Switzerland. **[6]**.

Yli-Jyrä, Anssi Mikael. 2004b. Coping with dependencies and word order or how to put Arthur's court into a castle. In H. Holmboe, ed., *Nordisk Sprogteknologi 2003. Nordic Language Technology 2003. Årbog for Nordisk Sprogteknologisk Forskningsprogram 2000–2004*, pages 123–137. Museum Tusculanums Forlag, Københavns Universitet.

Yli-Jyrä, Anssi Mikael. 2004c. A framework for reductions of automata recognizing restricted regular languages. Manuscript.

Yli-Jyrä, Anssi Mikael. 2004d. Simplification of intermediate results during intersection of multiple weighted automata. In M. Droste and H. Vogler, eds., *Weighted Automata: Theory and Applications, Dresden, Germany, June 1–5, 2004*, no. TUD-FI04-05 in Technische Berichte der Fakultät Informatik, ISSN-1430-211X, pages 46–48. D-01062 Dresden, Germany: Techniche Universität Dresden. **[8]**.

Yli-Jyrä, Anssi Mikael. 2005a. Approximating dependency grammars through intersection of regular languages. In *Implementation and Application of Automata, 9th International Conference, CIAA 2004. Kingston, Canada, July 22-24, 2004. Revised Selected Papers*, vol. 3317 of *LNCS*, pages 281–292. Springer-Verlag. **[3]**.

Yli-Jyrä, Anssi Mikael. 2005b. Approximating dependency grammars through intersection of star-free regular languages. Accepted for publication in *International Journal of Foundations*

of Computer Science.

Yli-Jyrä, Anssi Mikael. 2005c. Data vs. query complexity in treebank queries and induction of linguistic grammars. Poster presented at the 15th Nordic Conference of Computational Linguistics (NODALIDA 2005), University of Joensuu, Finland, May 20–21, 2005.

Yli-Jyrä, Anssi Mikael. submitted 2003. Regular approximations through labeled bracketing (revised version). Submitted to the post-proceedings of FGVienna, the 8th conference on Formal Grammar, Vienna, Austria, 16–17 August, 2003. According to the given information, the revised papers will be published as Online Proceedings by CSLI Publications, Stanford, CA, USA. **[2]**.

Yli-Jyrä, Anssi Mikael and Kimmo Koskenniemi. 2004. Compiling contextual restrictions on strings into finite-state automata. In L. Cleophas and B. W. Watson, eds., *The Eindhoven FASTAR Days, Proceedings*, no. 04/40 in Computer Science Reports. Eindhoven, The Netherlands: Technische Universiteit Eindhoven. **[7]**.

Yli-Jyrä, Anssi Mikael and Jyrki Niemi. 2005. An approach to specification of regular relations: Pivotal synchronization expressions. A full paper to be presented at the Finite-State Methods and Natural Language Processing 2005 (FSMNLP 2005), 5th workshop in the FSMNLP series, University of Helsinki, Finland, September, 2005.

Yli-Jyrä, Anssi Mikael and Matti Nykänen. 2004. A hierarchy of mildly context sensitive dependency grammars. In G. P. Gerhard Jäger, Paola Monachesi and S. Wintner, eds., *Proceedings of the 9th conference on Formal Grammar 2003 "FGNancy"*, pages 151–165. Preproceedings. Available at `http://cs.haifa.ac.il/~shuly/fg04/`. **[5]**.

Yu, Sheng. 1999. State complexity of regular languages (invited talk). In *Proceedings of Descriptional Complexity of Automata, Grammars and Related Structures*, pages 77–88.

Yu, Sheng and Qingyu Zhuang. 1991. On the state complexity of intersection of regular languages. *SIGACT News* 22(3):52–54.

Zeiger, H. Paul. 1968. Cascade decomposition of automata using covers. In M. A. Arbib, ed., *Algebraic Theory of Machines, Languages, and Semigroups*, pages 55–80. Netherlands: Academic Press.

List of Contributed Articles

Group 1

[1] Yli-Jyrä, Anssi Mikael. 2003a. Describing syntax with star-free regular expressions. In *11th EACL 2003, Proceedings of the Conference*, pages 379–386. Agro Hotel, Budapest, Hungary.

[2] Yli-Jyrä, Anssi Mikael. submitted 2003. Regular approximations through labeled bracketing (revised version). Submitted to the post-proceedings of FGVienna, the 8th conference on Formal Grammar, Vienna, Austria, 16–17 August, 2003. According to the given information, the revised papers will be published as Online Proceedings by CSLI Publications, Stanford, CA, USA.

[3] Yli-Jyrä, Anssi Mikael. 2005. Approximating dependency grammars through intersection of regular languages. In *Implementation and Application of Automata, 9th International Conference, CIAA 2004. Kingston, Canada, July 22-24, 2004. Revised Selected Papers*, vol. 3317 of *LNCS*, pages 281–292. Springer-Verlag.

Group 2

[4] Yli-Jyrä, Anssi Mikael. 2003b. Multiplanarity – a model for dependency structures in treebanks. In J. Nivre and E. Hinrichs, eds., *TLT 2003. Proceedings of the Second Workshop on Treebanks and Linguistic Theories*, vol. 9 of *Mathematical Modelling in Physics, Engineering and Cognitive Sciences*, pages 189–200. Växjö, Sweden: Växjö University Press.

[5] Yli-Jyrä, Anssi Mikael and Matti Nykänen. 2004. A hierarchy of mildly context sensitive dependency grammars. In G. P. Gerhard Jäger, Paola Monachesi and S. Wintner, eds., *Proceedings of the 9th conference on Formal Grammar 2003 "FGNancy"*, pages 151–165. Pre-proceedings. Available at `http://cs.haifa.ac.il/~shuly/fg04/`.

[6] Yli-Jyrä, Anssi Mikael. 2004a. Axiomatization of restricted non-projective dependency trees through finite-state constraints that analyse crossing bracketings. In G.-J. M. Kruijff and D. Duchier, eds., *Proc. Workshop of Recent Advances in Dependency Grammar*, pages 33–40. Geneva, Switzerland.

Group 3

[7] Yli-Jyrä, Anssi Mikael and Kimmo Koskenniemi. 2004. Compiling contextual restrictions on strings into finite-state automata. In L. Cleophas and B. W. Watson, eds., *The Eindhoven FASTAR Days, Proceedings*, no. 04/40 in Computer Science Reports. Eindhoven, The Netherlands: Technische Universiteit Eindhoven.

[8] Yli-Jyrä, Anssi Mikael. 2004b. Simplification of intermediate results during intersection of multiple weighted automata. In M. Droste and H. Vogler, eds., *Weighted Automata: Theory and Applications, Dresden, Germany, June 1–5, 2004*, no. TUD-FI04-05 in Technische Berichte der Fakultät Informatik, ISSN-1430-211X, pages 46–48. D-01062 Dresden, Germany: Techniche Universität Dresden.

[9] Yli-Jyrä, Anssi Mikael. 1995. Schematic finite-state intersection parsing. In K. Koskenniemi, ed., *Short Papers Presented at the 10th Nordic Conference of Computational Linguistics (NODALIDA-95)*, pages 95–103. Helsinki, Finland.

Errata for Contributed Articles

[1] Describing Syntax with Star-Free Regular Expressions

The first bullet under section 3.4 should start:

- The phrase "`every @>N @ 6,000 @>N @ miles N @ADVL`" satisfies ...

[4] Multiplanarity — a Model for Dependency Structures in Treebanks

Under the header *Planarity*, I made an attempt to define [semi]planarity by means of a logical restriction similar to the projectivity condition. The formally presented semi-planarity restriction in the article is, unfortunately, completely wrong[1]. First, we cannot make – as suggested – a transitive closure **governs*** symmetric by writing (**governs** $\cup$ **governs**$^{-1}$)*. The given formula is satisfied by a graph where all nodes govern each other. The semi-planarity condition for dependency graphs can still be expressed with **prec*** and **linked** as follows

> *if* $(A$ **linked** $B) \wedge (A$ **prec*** $B) \wedge (C$ **linked** $D) \wedge (C$ **prec*** $D) \wedge (A$ **prec*** $C)$
> then $(B$ **prec*** $C) \vee (D$ **prec*** $B)$.

(Note that we did not try to express planarity. It is well known that planarity is not definable in FO logic.)
Furthermore, the caption of first figure claims wrongly that the shown trees are *not planar*. Instead, they are planar but not projective. Finally, there is an inadequate reference to Wartena (2000). Instead, my intend was to make reference to Wartena (2001) where he defines *extended right-linear storage grammars*.

[5] A Hierarchy of Mildly Context-Sensitive Dependency Grammars

The pre-proceedings start numbering of definitions from 7 and theorems from 6. The numbering is not rational and we corrected it in the reprinted article. On page 161, the rule on the fourth line should read:

$$w(1/\overline{X}\ 1/V_1\ 1/V_2\ \ldots 1/V_n\ *\ 1/Y_1\ Y_2 \ldots Y_m), \text{ and}$$

Furthermore, on page 163, the 13th and 14th lines from the bottom: $\overleftarrow{\Gamma}$ and $\overrightarrow{\Gamma}$ should be swapped. Finally, on page 164, section 11.7 mixes definitions of dependency trees and projective dependency trees. It should read:

> Firstly, we start from the class of CNDGs, which guarantees acyclicity. Secondly, we require that all the rules of the forms (11.7) and (11.8) should contain at most one link (a link to a dependent), and that rules of the form (11.9) and (11.10)

[1] I am grateful to Marco Kuhlman for pointing out this silly error.

should contain exactly one governor link ($\rightarrow$ in-degree at most one). Thirdly, we assert that, in the complete derivations, the number of nodes that do not contain any governor links have to be exactly one ($\rightarrow$ connected graph). In this way we obtain grammars for non-projective dependency trees. Construction of HGDGs is on page 161.

[7] Compiling Contextual Restrictions into Finite State Automata

On the third page, under the formula (1), the condition for v and y should have read $v \in \mathcal{V}_i$ and $y \in \mathcal{Y}_i$ because we intended to define each $\mathcal{V}_i$ and $\mathcal{Y}_i$ as total contexts. On page 13, the last column of table 1 contains errors. The column should contain the items: 3^d12 (header), 12, 36, 108, 324, 972.

[8] Simplification of intermediate results during intersection of multiple weighted automata

Item [4] in the list of references should read:

P. Tapanainen. Applying a finite-state intersection grammar. In E. Roche and Y. Schabes, editors, *Finite-state language processing*, pages 311–327. A Bradford Book, MIT Press, Cambridge, MA, 1997.

[9] Schematic Finite-State Intersection Parsing

In the end of section 4, the last occurrence of σ (right above the figure 10) should have been printed as ρ.

The up-to-date list of corrections will be available on the WWW page devoted to this dissertation. Currently it is located at

http://www.ling.helsinki.fi/~aylijyra/dissertation/.

Index of Acronyms