Parsing in two frameworks: finite-state and functional dependency grammar

Pasi Tapanainen
University of Helsinki, FIN

December 1st, 1999

ISBN 951-45-8752-9 (PDF version)
Helsinki 1999
Helsingin yliopiston verkkojulkaisut

Contents

1 Introduction

The present PhD thesis consists of the following articles:

1. Tapanainen, Pasi and Timo Järvinen. 1997. A non-projective dependency parser. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, pages 64–71, Washington, D.C., April. Association for Computational Linguistics.

2. Tapanainen, Pasi. 1997. Applying a finite-state intersection grammar. In Emmanuel Roche and Yves Schabes, editors, *Finite-state language processing*, A Bradford Book. MIT Press, Cambridge, Massachusetts, chapter 10, pages 311–327.

3. Tapanainen, Pasi and Timo Järvinen. 1998. Dependency concordances. *International Journal of Lexicography*, 11(3):187–203, September.

4. Chanod, Jean-Pierre and Pasi Tapanainen. 1999. Finite state based reductionist parsing for French. In András Kornai, editor, *Extended Finite State Models of Language*, Studies in Natural Language Processing. Cambridge University Press, chapter 8, pages 72–85.

This summary describes a group of parsing formalisms which are based on disambiguation in terms of regular languages. The articles Tapanainen (1997) and Chanod and Tapanainen (1999) describe this kind of reductionist parsers. The articles Tapanainen and Järvinen (1997; 1998) present a novel parsing framework, *Functional Dependency Grammar* (FDG). FDG combines two inverse approaches: a regular language based reductionist approach, which disambiguates the linear, surface structure of the sentence by discarding readings of tokens, and a tree based dependency model, which creates mutual dependencies between syntactic elements.

In the regular language based systems, a reduced linguistic structure is represented using a tagging scheme, such as the one in Voutilainen (1997), where every word has a tag or tags describing the properties of the word or its implicit relations to the words appearing in the left or right hand context. The dependency graphs, however, are based on the relations between the elements of the sentence. In the original representation (Tesnière, 1959), the order of the words in the sentence is not present in the linguistic analysis.

In working with the Functional Dependency Grammar, we have adopted, formalised and developed the structural syntax of Tesnière

(1959). This is discussed in more detail in Järvinen and Tapanainen (1998) and Järvinen (1998).

My original contribution to Tapanainen and Järvinen (1997), Tapanainen and Järvinen (1998) and Chanod and Tapanainen (1999) is in the design of the parsing systems and their formalisms, whereas the linguistic descriptions are by my coauthors, Timo Järvinen and Jean-Pierre Chanod.

In Tapanainen (1997), I present methods of applying rules in a finite-state based rule formalism. I first show that the naïve approach to apply finite-state rules in Koskenniemi (1990) is infeasible, and then present methods of applying the rules. I have designed these methods when I worked at the Research Unit for Computational Linguistics (RUCL) at the University of Helsinki.

My main contribution in Tapanainen and Järvinen (1997), as mentioned above, is in the new type of parsing formalism and the engine. Timo Järvinen wrote the grammar for English. We show that the parser is fast and more accurate than some other state-of-the-art parsers. The much shallower ENGCG syntax (Järvinen, 1994), especially, creates four times the ambiguity and makes more errors than FDG, when measured at the level of syntactic functions (tags) attached to words.

In Tapanainen and Järvinen (1998), we discuss an application of the parser. I implemented a novel approach to use a syntactic parser in lexicographic work and we show that this approach is practical. I did this work at the Research Unit for Multilingual Language Technology (RUMLAT) at the University of Helsinki.

In Chanod and Tapanainen (1999), we present a paradigm where all the tasks from the tokenisation to the syntactic analysis apply finite-state automata and transducers. My contribution to this article is in:

- the novel non-determistic tokenisation method which was first presented in Tapanainen (1995) and Chanod and Tapanainen (1996a),
- the formalism for presenting multiword units which was first presented in Tapanainen (1995) and Segond and Tapanainen (1995),
- the combination of the tokenisation, multiword unit recognition, lexical analysis and syntactic analysis, and
- the syntactic disambiguation engine which is similar to that in Tapanainen (1997).

I did this work when working in the Multilingual Language Technology and Theory group at the Xerox Research Centre in Grenoble.

This thesis represents the whole process from tokenisation to surface syntactic analysis. In Chanod and Tapanainen (1999), we present the syntactic language analysis from tokenisation to light syntax, and in Tapanainen (1997), I present more detailed methods for applying regular language based syntactic grammars. In Tapanainen and Järvinen (1997; 1998), we discuss explicit dependency structures with more fine-grained surface syntactic analysis.

1.1 Surface syntactic analysis

My purpose in this dissertation is to analyse language in terms of surface syntactic structures. There are several layers of language information available. The following exposition is a condensation of Chanod and Tapanainen (1999), where we discuss the tasks of natural language analysis in terms of regular language calculus, and Tapanainen and Järvinen (1997), where we discuss structural syntactic analysis.

- The first task is *tokenisation*. It segments the continuous input stream into tokens. This is generally considered an independent step in language analysis, but we integrated it with the other modules because in syntactic analysis the tokens should be syntactic elements rather than orthographic elements like words. Selecting a correct token in a complex syntactic position is hard and generally requires that syntactic analysis is already at least partly done. Thus the tokeniser produces ambiguous tokens, e.g. the string *de même* can be one token ("similarly") or two tokens ("of" and "same").

- *Lexical analysis* provides each token with a set of labels denoting different word-classes, morphological properties and syntactic information. These labels are typically extracted from a dictionary or presumed at by using the form of the token. *Morphological analysis*, which resolves the internal parts of word forms, is a typical form of lexical analysis. Figure 1 shows the result of morphological analysis for the English word *round*. The result is ambiguous because the word *round* is either a noun (N), adjective (A), preposition (PREP), adverb (ADV) or verb (V).

- A *multiword unit* consists of one or more words or word parts, and appears in the parsing system as one or more tokens, but the number of tokens does not depend on the number of words in the input. For instance, the string *il va y avoir bientôt cinq ans* ("five years ago")

is encoded as a single token (an adverb), whereas the string *du* is divided into two tokens: *de* + *le* (preposition + determiner).

The definition language can describe one-to-one correspondences with words and tokens, but restricts the alternative analyses for the tokens. For instance, the following declaration

```
:Afrique :noire Adj:
```

means that the token *noire* ("black") receives only the adjective reading in this context. Multiword unit recognition is related to and integrated with tokenisation and lexical analysis.

- *Syntactic function labels* mark various syntactic properties. Such labels mark, for example, subjects, objects, main verbs and various complements. Alternative syntactic labels are introduced in the lexical analysis or at run-time before syntactic disambiguation.

- *Morpho-syntactic disambiguation* discards ambiguous labels produced by the previous steps. Some regular language based disambiguation methods are discussed in more detail in Tapanainen (1997). In Figure 2, there is an analysis from the surface syntactic parser of French (Chanod and Tapanainen, 1999) for the sentence *De même les boîtes de même format sont classées ensemble* (literally: "Similarly the boxes of (the) same format are classified together.").

- *Surface syntactic dependency structures* are presented as graphs. I am focusing on dependency based models and structures from Tapanainen and Järvinen (1997; 1998).

- A *nucleus* is the basic element of the syntactic structure in the dependency model. The nucleus has the same task as a multiword unit but it is more general. Whereas a multiword unit consists of a sequence of words or word parts, a nucleus consists of tokens which syntactically and semantically belong together. Thus the nucleus is often a non-contiguous sequence of words or word parts without internal syntactic structure.

```
round
    "round" N NOM SC
    "round" A ABS
    "round" ADV ADWL
    "round" PREP
    "round" V PRES -SG3 VFIN
    "round" V INF
    "round" V IMP VFIN
    "round" V SUBJUNCTIVE VFIN
```

Figure 1: Output of a morphological analyser

de-même	Adv Cap MWE	**Adverbial**	./
le	InvGen PL Def Det	**NounPrMod**	./
boîte	Fem PL Noun	**Subject**	./
de		**Prep**	./
même	InvGen SG Adj	**NounPrMod**	./
format	Masc SG Noun	**PPObj**	./
être	IndP PL P3 Verb Copl Auxi	**MainV**	./
classer	PaPrt Fem PL Verb	**PastPart**	./
ensemble	Adv	**Adverbial**	_.

Figure 2: Output of a surface syntactic parser

1.2 Related work

Deterministic tokenisation, as opposed to non-deterministic tokenisation (Chanod and Tapanainen, 1999), is discussed in Grefenstette and Tapanainen (1994), Palmer and Hearst (1994) and Karttunen et al. (1997). The type of morphological analysis that we used is based on the two-level model (Koskenniemi, 1983) and similar, more advanced transducer lexicons (Karttunen, Kaplan, and Zaenen, 1992; Karttunen, 1994). The multiword recognisation formalism is described in Segond and Tapanainen (1995) and its compiler in Tapanainen (1995).

The use of regular language based systems for natural language processing has a long history. The first system that I am aware of was designed by Harris (1962), and essentially contained, according to Joshi and

Hopely (1999), a cascade of finite-state transducers. A later, more closely related natural language analysis system is Taggit by Greene and Rubin (1971), which was not originally described in terms of regular language.

In Harris' parser, each transducer has a specific task from making word compounds to disambiguation and bracketing phrases by using the left or right context. In this way, it resembles chunking by Abney (1991). Chunking was later combined with finite-state intersection grammar (Section 2.4) by Aït-Mokhtar and Chanod (1997). Another derivation from the finite-state intersection grammars is a tagging system for Turkish by Oflazer and Kuruöz (1994). Some other finite-state based approaches are presented in Roche and Schabes (1997), Karttunen and Oflazer (1998) and Kornai (1999).

In Tapanainen and Järvinen (1997) and Järvinen and Tapanainen (1997), we adopted the basic features of our linguistic dependency model from Tesnière (1959). These features contain the notions of dependency, valency and nucleus that have a faithful implementation in our model. See Järvinen (1998) for further discussion. The closest relatives to our dependency model come from Prague. The Functional Generative Description (Sgall, Hajičová, and Panevová, 1986; Hajičová, 1998) is a linguistic model which describes "systemic order" (a kind of deep structure) of sentences. Another close relative is dependency syntax by Mel'čuk (1987).

Heringer (1993) presents some other related models. One well-known model comes from Hudson (1984; 1991). It differs in some crucial respects, such as in the definition of projectivity and multiple heads which are not allowed for in our framework. Fraser (1989) made a partial implementation of Hudson's dependency theory, but a successful, related parser comes from McCord (1990). Järvinen (1998) describes the linguistic similarities and differences between various dependency models and ours in more detail.

There are some early formalisations of dependency theory by Gaifman (1965) and Hays (1964), that were defined in the terms analogical to constituency (context-free) grammars. The Link grammar of Sleator and Temperley (1991) is a lexical dependency based parser which has little functional information. Lin (1996) has a dependency based evaluation scheme for his Principar parser. There is a dependency parser for Czech (Holan, Kuboň, and Plátek, 1997) and for French (Giguet and Vergne, 1997). There has also been research on fully lexical dependency systems which mainly produce only head-modifier information without any functional information. These are typically probabilistic systems like those in Eisner (1996) and Collins (1996). Some other dependency oriented approaches are presented in Kahane and Polguère (1998). Some dependency based treebanks

are presented for Czech in Hajič (1998), for Japanese in Lepage et al. (1998) and for German in Skut et al. (1997).

The closely related parsers, such as Taggit (Greene and Rubin, 1971), Constraint Grammar (CG-1) (Karlsson, 1990) with its application, ENGCG (Voutilainen, Heikkilä, and Anttila, 1992; Karlsson et al., 1995), and Constraint Grammar CG-2 (Tapanainen, 1996), are described in Section 2. There are a few constraint grammar (CG-2) based linguistic descriptions. Such grammars are written, e.g. for English (Samuelsson and Voutilainen, 1997), Swahili (Hurskainen, 1996), Portuguese (Bick, 1997) and Basque (Aduriz et al., 1996).

There are experiments with constraint grammar acquisition. In Tapanainen and Järvinen (1994), we built a system which collected finite-state automata which best describe the partially ambiguous learning material. Procedures for acquisition of constraint grammar rules are described in Samuelsson, Tapanainen, and Voutilainen (1996) and later in Lindberg and Eineborg (1998). Constraint grammars were combined with a statistical model in Tapanainen and Voutilainen (1994) and they are even presented as a statistical model, as in Brants and Samuelsson (1995).

1.3 Acknowledgment

Timo Järvinen (1998) wrote the first linguistic description (English) in the Functional Dependency Grammar framework.

The linguistic descriptions for various regular language based systems presented in this summary were written by **Jean-Pierre Chanod** (Chanod and Tapanainen, 1995; Chanod and Tapanainen, 1999) for French, and **Atro Voutilainen** (1997) for English. The morphological disambiguation grammar for the CG-1 type of constraint grammar for English, ENGCG, was made by Voutilainen (1994; 1995) and the related syntactic disambiguation grammar by **Arto Anttila** (1995) and Järvinen (1994). The CG-2 variation of the morphological disambiguation grammar for English comes from Voutilainen.

I am indebted to Jean-Pierre Chanod, Timo Järvinen and Atro Voutilainen for valuable discussion and feedback in my work, especially in the development of the parsing frameworks.

I am indebted to Timo Järvinen, **Fred Karlsson**, **Kimmo Koskenniemi**, **Jussi Piitulainen** and Atro Voutilainen for their critical comments on earlier versions of this manuscript.

2 Finite-state constraint languages

Definition: A finite-state constraint language is a language for specifying a disambiguation machine that discards alternative readings of syntactic elements by contextual tests that are expressed in terms of regular languages.

I made several different engines for constraint grammars. The first versions (Tapanainen, 1991; Tapanainen, 1992; Tapanainen, 1997) are the so-called intersection grammars that I implemented at the University of Helsinki. Their rule application engine is defined by the intersection operation of regular languages. The later ones (Tapanainen, 1995) were built at Rank Xerox Research Centre in Grenoble: one is an engine for a tagger by Chanod and Tapanainen (1994; 1995), having features from both the ENGCG-style constraint grammars (Karlsson et al., 1995) and intersection grammars. Another one is applied in a light syntactic parser by Chanod and Tapanainen (1996b; 1999).

The CG-1 type of engines are mutually slightly different in their expression power. So far, their expression power is less than regular languages. The first CG-1 type of engine I made is quite faithful to the morphological and syntactic disambiguation part of ENGCG. This work was later continued by defining and implementing CG-2 (Tapanainen, 1996), whose formalism is more expressive than that of CG-1, if not quite that of regular languages.

2.1 Taggit

The Taggit program (Greene and Rubin, 1971) was used for annotating the Brown Corpus (Francis and Kučera, 1982). Like any parsing program, Taggit contains several steps starting from preprocessing and lexical analysis, but the most interesting part in Taggit is the *Context Frame Test*, which closely resembles our definition of a finite-state constraint language. Greene and Rubin (1971, p. 2) describe it as:

> After the words of a sentence have been given all their possible tags, Context Frame Tests are applied to ambiguities to resolve them if possible. Basically, a context test decides what tag(s) can or cannot occur in a particular unambiguous context. These rules may be positive, choosing one tag from among two or more, or negative, elimiting one or more tags.

The basic idea of removing ambiguity is the same as in the later Constraint Grammar framework. The context tests in Taggit refer to one or two po-

sitions, left or right. It should be noted that the formalism that Taggit applied was not very powerful, but a tiny subset of regular languages.

A typical disambiguation rule in Taggit is

AT 1 → ¬VB

which means that an ambiguous token in position 1 (i.e. the following word) loses (¬) the verb reading (VB) after an unambiguous article (AT).

2.2 Constraint Grammar CG-1

In Karlsson's (1990; 1995) Constraint Grammar framework, there are seven modules for different tasks. Two of them, i.e. morphological disambiguation and syntactic function disambiguation, can be expressed with finite-state constraint languages. I call this type of disambiguation engine CG-1.

The basic operations in this model are `=0` (discard a reading), `=!` (discard all other readings) and `=!!` (discard a reading or all other readings depending on the success of the tests). A typical morphological disambiguation rule is

```
(@w =0 (V) (-1C DET)).
```

It discards the verb reading (V) if the previous token is unambiguously a determiner (DET).

The rules can refer to any position in a sentence, i.e. there is no other "window" than the sentence itself. The means to refer to distant tokens are restricted though.

The first application in the CG-1 framework is ENGCG (Voutilainen, Heikkilä, and Anttila, 1992; Karlsson et al., 1995). There are three implementations of the engine listed in Karlsson et al. (1995, p. 45): lisp version by Fred Karlsson (1990), one C++ version by Bart Jongejan, CRI A/S, (created in 1992) and one C version by myself (created in 1993), which was not based on the original CG-1 description but rather on the most essential requirements of ENGCG.

2.3 Constraint Grammar CG-2

CG-2 is an independent revision of constraint grammar. I designed (Tapanainen, 1996) both a new engine and a new formalism. The objective in developing the formalism was to make it readable and efficient to apply. The contextual tests gained more power in CG-2 compared with the older CG-1. The basic rule types in the formalism are the same as in Taggit and

CG-1, namely *SELECT* and *REMOVE*. A minor difference here is that the SELECT rule can simultaneously select a group of readings, while both Taggit and CG-1 always select one reading.

A rule for removing the verb readings after an unambiguous determiner is the following:

```
REMOVE (V)
   IF (-1C DET);
```

A generalised form of this rule is: remove verb readings after an unambiguous determiner if there are no noun phrase head candidates between:

```
REMOVE (V)
   IF (*-1C DET BARRIER NPHEAD);
```

where NPHEAD is a barrier which prevents the testing outside of potential noun phrase heads.

The CG-2 based morphological disambiguation of English is evaluated in Samuelsson and Voutilainen (1997).

2.4 Finite-state intersection grammar

The basic idea in finite-state intersection grammar (FSIG) (Koskenniemi, 1990; Koskenniemi, Tapanainen, and Voutilainen, 1992) is that a sentence with all its ambiguity is represented as a finite-state automaton, so that every path from the start state to a final state of the automaton represents one alternative analysis for the sentence. The labels on the path represent the tokens, word-class markers, syntactic functions and clause boundaries. A rule is similar to the morphological rules for two-level morphological analysis. It is then compiled into a finite-state automaton. The automata generated from the rules are intersected with the automaton generated from the sentence. The set of strings accepted by the intersection represents the set of correct analyses. One such string is presented in Figure 2. The similarity to the morphological analysis tools is noted by Karttunen (1994), who applies morphological rules to a large dictionary by using a very similar method as in the first operating analyser for the intersection grammars (Tapanainen, 1991; Koskenniemi, Tapanainen, and Voutilainen, 1992).

The general form (Chanod and Tapanainen, 1999) of a rule in finite-state intersection grammars is

$$X \Rightarrow LC_1 \_ RC_1, LC_2 \_ RC_2, \ldots, LC_n \_ RC_n ;$$

This means that any string accepted by the regular expression X has to appear between regular expressions LC_1 and RC_1, or between LC_2 and RC_2, and so on.

My work with finite-state intersection grammars is presented in Tapanainen (Tapanainen, 1991; Tapanainen, 1992; Tapanainen, 1997), Koskenniemi, Tapanainen, and Voutilainen (1992), Voutilainen and Tapanainen (1993), Tapanainen and Voutilainen (1993), Chanod and Tapanainen (1995; 1996b; 1999). In Tapanainen (1997), I present several alternative approaches for resolving the intersection of finite-state automata in this framework. Some other approaches are presented in Piitulainen (1995) and Yli-Jyrä (1995).

2.5 Application order of rules

A common feature for CG-1 and CG-2 is that the rules are applied in order, but the order of application is due to the engine, and different engines have different preferences for the application order. The first CG-1 engines tried to apply the rules in order. In CG-2, the grammarian who writes the rules has minor chances to control the order. The consequence is that the rules have to be carefully written and a reading should be discarded only in a carefully certified context. A grammarian applies a "Sherlock Holmes strategy": one can not always straightforwardly remove a reading, but by using available evidence one can discard a reading here and there as soon as they appear illegitimate. The last reading is never removed. Finally, hopefully, the hard ambiguity can be resolved.

Most implemented engines for finite-state constraint languages have a possibility for rough categorization by grouping the rules. The rules are divided in Taggit into reliable and less reliable rules. The program adds an asterisk into the tokens which are disambiguated using less reliable rules so that the errors would be easier to find. CG-1 has a similar categorization, which Karlsson (1990) calls disambiguation rules and heuristic disambiguation rules. The number of the rule sections in CG-2 is not limited, which makes it possible to revise the purpose of the sections. The major purpose should not be to divide rules into reliable and less reliable ones, but to give a function to the different sections. One functional way to divide the rules is proposed in Chanod and Tapanainen (1995): The first section resolves idiomatic and exceptional usage of language which does not follow the core syntactic description. The second section contains the standard rules to which most of the target language follows, and finally, the third section disambiguates remaining ambiguous tokens by

removing lexically rare analyses. Furthermore, blindly dividing the rules according to how reliable they are in certain large text corpora prevents an improbable usage ever being analysed correctly, even when the syntactic construction is fully recoverable by syntactic rules.

2.6 Expression power

The rules of Taggit can be expressed with the following two types of CG-2 rules:

```
SELECT (A)
   IF (-2C T1)
      (-1C T2)
      (1C T3)
      (2C T4);
```

and

```
REMOVE (A)
   IF (-2C T1)
      (-1C T2)
      (1C T3)
      (2C T4);
```

where any of the tests can be missing.

The regular expression '$(\Sigma - B)^* D (\Sigma - E)^* F$', for instance, can be expressed as

```
*1 D BARRIER B LINK *1 F BARRIER E
```

in CG-2, but it can not be expressed with a rule of the CG-1 formalism.

There is no convenient way in the current implementation of CG-2 to refer to repeated patterns such as the regular expression '$(D\ A\ N)+$' where the sequence of the labels D, A and N is repeated any number of times. This, on the other hand, is possible in the implemented intersection grammar (Tapanainen, 1997), which accepts a wide range of regular language operations, including concatenation, Kleene's closures, union, negation and intersection. This makes this intersection grammar equivalent to regular languages. Figure 3 shows the hierarchy of the expression power of the implemented engines for finite-state constraint languages.

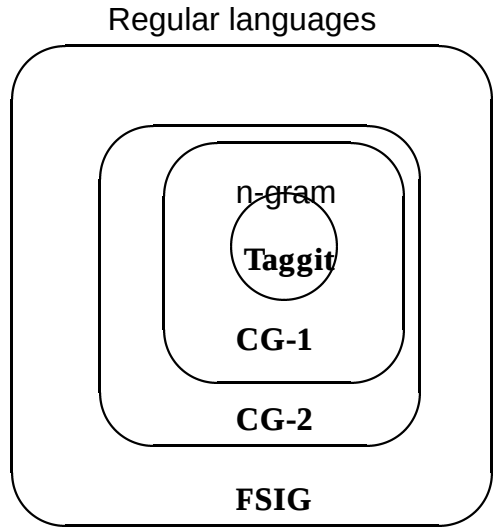

Figure 3: Expression power of implemented constraint language engines

2.7 Time requirement

For the time requirement, I approximate both the theoretical worst case asymptotic running time of constraint grammars and the average speed of CG-2, which is the most efficient implementation.

The three CG-1 engines for ENGCG have huge differences in speed. Karlsson (1995, p. 45) lists the speeds in Sun SparcStations as following: Karlsson (lisp program) 3-5 words per second, Jongejan (C++ program) 15-20 per second and Tapanainen (C program) 400-500 per second. The speed noted is for the whole process including preprocessing, morphological disambiguation, syntactic disambiguation, etc. While the speed of CG-2 can not be mechanically compared with the CG-1 engines because of incompatible rule formalisms, there is a case when the ENGCG disambiguation description (Voutilainen, 1995) was converted (Tapanainen, 1996) from the CG-1 formalism into the CG-2 formalism. The first CG-2 implementation was two times faster than the fastest CG-1 engine above. Currently, the CG-2 runs much faster, and in a current Pentium II machine it analyses, with several thousands of rules and integrated lexical analysis, over 100 MB of text in an hour.

2.7.1 Worst case asymptotic running time of CG-2

Theorem: The worst case asymptotic running time for the CG-2 type of constraint grammars is $O(n^3Gk^2)$, where n is the length of the sentence,

G the number of the rules in the grammar and k the maximal number of different readings a token receives.
Proof: Because constraint grammars use regular languages, the tests of a rule can be made in linear time $O(nk)$ to resolve whether a rule should discard or select an alternative reading of a token. It takes time $O(nGk)$ to test all the rules to decide which readings to discard. It takes time $O(n^2Gk)$ to test all the tokens.

In the worst case, there is only one rule which can remove only one reading in the sentence, and it is the last one tested. The context of the other rules then changes, and some of the other rules may apply. In the worst case, again, there is only one rule which can discard only one reading. Thus, it takes $n(k-1)$ rounds to make fully disambiguated output, and the time needed is $O(n^3Gk^2)$.

Usually, the maximal ambiguity class is a constant. If we consider the somewhat outdated English description in the old ENGCG, the lexicon produces eight lexical readings at most and there are some 30 syntactic functions. This makes the upper limit for k 240. In a hypothetical case, it might be that a very liberal compound word mechanism produces ambiguity for each partial word in a compound word. This then cumulates into a massive amount of ambiguity for a theoretical, very long compound word. On the other hand, this ambiguity is restricted by the size of the tag set in the grammar, because the constraint grammar can not remove such readings one by one for which it does not have rules. If we redefine the size of the grammar G' to be the amount of tag combinations in the grammar, we thus have the asymptotic time bound $O(n^3G'^2)$, because $G < G'$ holds. This is similar to context-free (constituent) grammars.

2.7.2 Average running time of CG-2

The average running time is tested on CG-2. Unfortunately, it is hard to reliably approximate the asymptotic time because there are not sentences of unbounded length in natural language. It is even questionable if the long sentences in the samples should be considered real sentences rather than list constructions.

I used the following testing material: 3 500 rules from Atro Voutilainen's disambiguation grammar for English and two texts: a selection of novels taken from the Gutenberg archives from the 1997 distribution (some 1 000 000 sentences) and a sample of the *Financial Times* from the TREC Text Research Collection, Vol. 4 CD-ROM (some 5 000 000 sentences). The samples were run in a 133 Mhz Pentium PC.

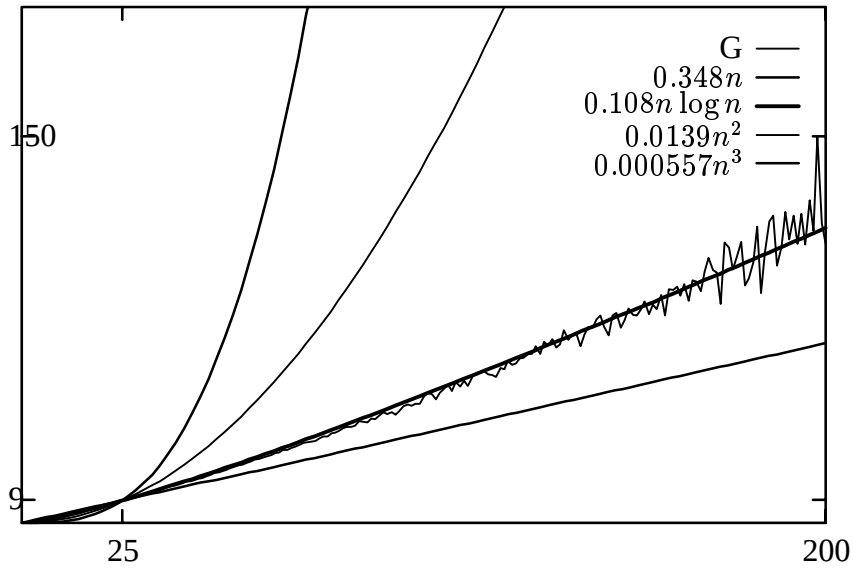

Figure 4: Average running time of CG-2 in Gutenberg texts

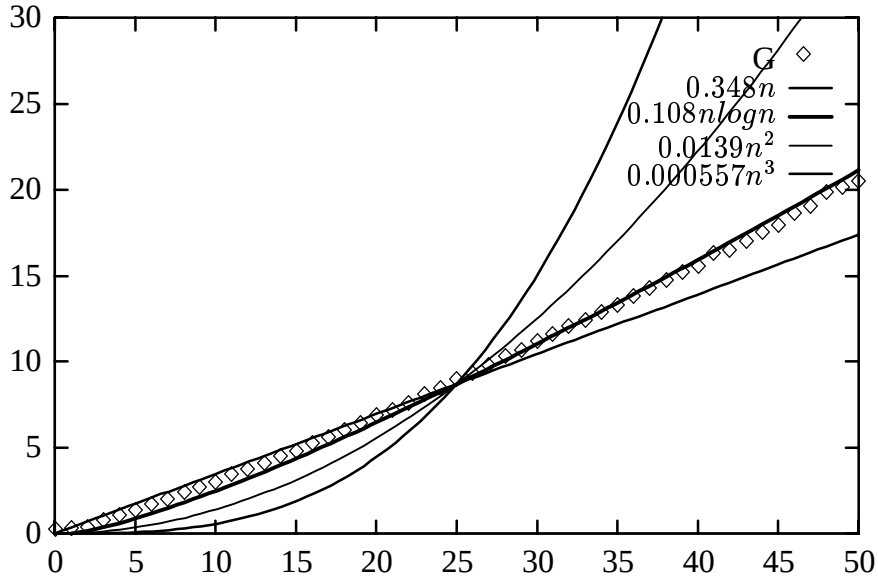

Figure 5: Average running time of CG-2 in Gutenberg texts in sentences with less than 50 tokens

The method was the following: the texts were disambiguated by CG-2, which printed the running time for each sentence. The running times were clustered: the average time for each sentence length is counted. If this cluster is too small, the cluster is combined with a neighbouring cluster.

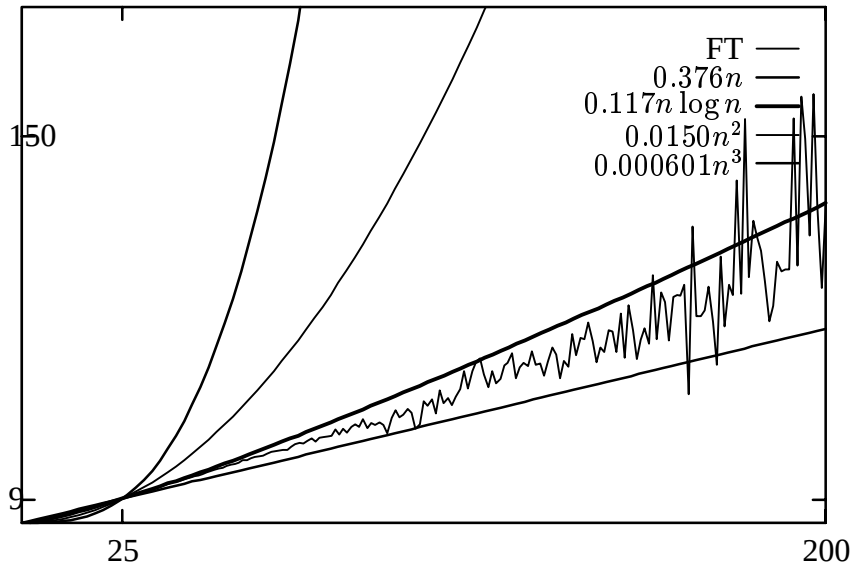

Figure 6: Average running times of CG-2 in the *Financial Times*

The average running time is then plotted in Figures 4 to 6. In addition to the running time, there are four function curves: linear (n), quadratic (n^2), cubic (n^3) and $n \log n$. I set a coefficient for all of these functions so that they go through the same point which denotes the sentence length 25. The time in the y axis is in milliseconds.

The running time curve from parsing the novels seems smooth in Figure 4, closely following the $O(n \log n)$ curve. Figure 5 shows the curves for the short sentences in more detail. On the other hand, the curve of the newspaper text in Figure 6 seems somewhat more complex. The newspaper obviously has a larger variation in the running time. Nevertheless, the $O(n \log n)$ time seems a reasonable approximation for the average running time of CG-2 in both cases.

2.7.3 Worst case asymptotic running time of intersection grammars

The theoretical worst case running time of the intersection grammars is discussed in Tapanainen (1997). There, I showed that this type of engine runs in linear time $O(GS)$, where G is the size of the combined compiled grammar and S is the size of the sentence compiled into a finite-state automaton. The size S is linear to the length of the sentence if the amount of ambiguity that an individual token may get is limited. Paradoxically, Voutilainen (1998) reports that due to the massive computation needed with

his 2 600 rule grammar and my engine (Tapanainen, 1992), the engine is unable to find a correct parse for some long sentences which have such a parse. This seems to contradict the expectation that a linear upper bound guarantees efficient computation.

There is another calculation in Tapanainen (1997) for the worst case asymptotic time where the running time is computed without any knowledge on the grammar. It is shown that the time needed may have exponential growth when compared with the number of ambiguous tokens, w, in the sentence.

The explanation of the paradox is that while the worst case is linear with a fixed grammar, the coefficient G is huge. The size of the intersection of two finite-state automata is in the worst case nm, where n and m are the sizes of the automata. Unfortunately, the pairwise tests on the rules of the existing grammar show that this happens often.

3 Functional Dependency Grammar

The Functional Dependency Grammar (FDG) framework was introduced by Tapanainen and Järvinen (1997) and its linguistic basis is discussed in Järvinen and Tapanainen (1997; 1998) and in Järvinen (1998). Some of its applications are presented in Tapanainen and Järvinen (1998), Tapanainen, Piitulainen, and Järvinen (1998) and Strzalkowski et al. (1999).

The Functional Dependency Grammar has two conceptually different components: regular language based analysis and structural dependency graphs. The linguistic representation in structural dependency graphs closely follows the Structural Syntax of Tesnière (1959). To my best knowledge, FDG is the first computer implementation of Tesnière's Dependency Theory which contains the essential ideas of the theory, namely:

- The basic element of the syntactic structure is not a word but the nucleus.
- Nuclei of the sentence have mutual directed dependencies called connexions.
- Every nucleus has one and only one syntactic head. The connexions thus form a tree where the head element of the main clause is the root of the sentence.
- The variation in word order in the sentence does not affect the structural analysis when the syntactic function of the words remain the same.
- There is a close parallelism between syntax and semantics, i.e. the syntactic structure is motivated by the semantic interpretation rather than by word-order configurations, morphological markings or historical descriptive practice.

Tesnière has been largely ignored and misunderstood in literature as has been noted by Engel (1996). There are certain formalisations, such as in Gaifman (1965) and Hays (1964), that were claimed by Robinson (1970) to be formalisations of Tesnière's (1959) theory but have little in common with it. This is discussed in more detail in Järvinen and Tapanainen (1998).

Because of the different nature of the components of FDG, the components represent syntactic information in distinctive ways. The shallow, constraint grammar type of analysis for the sentence, *What would you like*

What	“what”	@OBJ PRON
would	“would”	@+FAUXV V
you	“you”	@SUBJ PRON
like	“like”	@-FMAINV V
me	“i”	@OBJ PRON
to	“to”	@INFMARK> INFMARK>
do	“do”	@-FMAINV V
?		

Figure 7: Shallow analysis

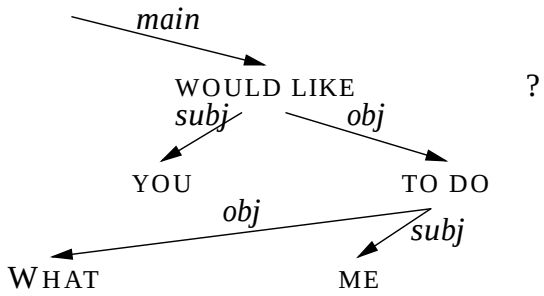

Figure 8: Structural dependency graph

me to do?, is presented in Figure 7. This analysis is a simple string of labels, where the labels are interpreted as words, word-classes and surface-syntactic tags. The structural dependency graph of the sentence in Figure 8 shows the directed relations between the nuclei of the sentence. The directed relations contain labels which represent syntactic functions between the governing node and the subtree.

The example sentence demonstrates the independence of the components of FDG. First, the tokens and nuclei do not have one-to-one mapping. Here two non-contiguous tokens, namely *would* and *like*, form a single nucleus in the syntactic graph. Second, the linguistic interpretation of the tokens and nuclei are, in principle, different. The token *me* is marked as *@OBJ* (object) in the shallow analysis, which is mostly morphologically based. In the structural graph, which is semantically motivated, the similar element, e.g. the nucleus *me* has the *subj* (subject) relation with its head, the nucleus *to do*. This example shows how the two linguistic interpretations are independent from each other.

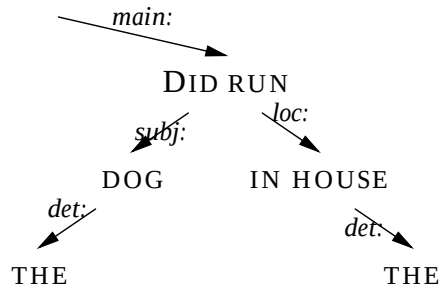

Figure 9: Five nuclei of the sentence “Did the dog run in the house”

3.1 Non-contiguous nucleus

The basic element of the structural analysis is called a *nucleus* (Tesnière, 1959). In FDG, the nucleus consists of tokens, which are defined by the lexicon and the tokeniser of the regular language model, where the token strings consist words and word parts in the input sentence.

In FDG, we have a nucleus predicate n, which is a reflexive symmetric predicate of tokens T in the regular language model. Let n be a nucleus predicate. The predicate n is symmetric and thus it holds $n(t_1, t_2) \in n \Leftrightarrow n(t_2, t_1) \in n$. The nucleus predicate is reflexive, which means that $n(t_i, t_i) \in n$ holds for all $t_i \in T$.

Definition: Tokens t_1 and t_n belong to the same nucleus in the nucleus predicate n if there are tokens $t_1, t_2, \ldots t_n$, where it holds $n \geq 1$ and $n(t_i, t_{i+1})$ for all $i \in [1, n-1]$.

For example, let S be the sentence *Did the*$_1$ *dog run in the*$_2$ *house* and $\{n(\text{Did}, \text{run}), n(\text{in}, \text{house})\}$ a set of nucleus predicates. Thus, the whole set of the nucleus predicates is:

$$\begin{aligned} N = \{ & n(\text{Did}, \text{run}), n(\text{run}, \text{Did}), n(\text{in}, \text{house}), n(\text{house}, \text{in}), \\ & n(\text{Did}, \text{Did}), n(\text{the}_1, \text{the}_1), n(\text{dog}, \text{dog}), n(\text{run}, \text{run}), \\ & n(\text{in}, \text{in}), n(\text{the}_2, \text{the}_2), n(\text{house}, \text{house})\}. \end{aligned}$$

The nuclei of the sentence thus contain the following sets of tokens: *the*$_1$, *dog, Did run, the*$_2$ and *in house* as presented in Figure 9.

3.2 Structural analysis

Definition: The dependency predicates are triples $\delta(n_1, n_2, f)$, where $n_1 \in N \cup \{root\}$ (head) and $n_2 \in N$ (modifier) belong to the nuclei (N) of a

sentence or n_1 is the special nucleus called the *root*, and $f \in F$ belongs to a set of labels denoting the functions in a selected linguistic representation.
Definition: The predicate $\delta^*(n_1, n_k, f)$ is the reflexive transitive closure of the dependency predicate δ if there are nuclei $n_1, n_2, \ldots n_k$, where it holds $k \geq 1$ and $\delta(n_i, n_{i+1}, f)$ for all $i \in [1, k-1]$.

Furthermore, we selected a linguistic theory with two uniqueness principles. First, the so-called head nucleus n_1 is unique to each modifier n_2, i.e. it holds $\exists x, z \in N \cup \{root\} : y \in N : \delta(x, y, f_1) \wedge \delta(z, y, f_2) \Rightarrow x = z \wedge f_1 = f_2$. Second, we divide the set of predicates $F = F_v \cup F_a$ into two categories: valency functions F_v and ambiguous functions F_a. The valency functions are unique in every head nucleus, i.e. it holds $\exists x, y, z \in N : \delta(x, y, f) \wedge \delta(x, z, f) \Rightarrow z = y$. Both of these principles originate from Tesnière (1959).

It should be noted that Tesnière (1959) does not describe the model with formal rules, but rather describes it verbally and via examples. Although he maintains the uniqueness of the head, he does not require the uniqueness principle simultaneusly to hold over all levels in natural language analysis. He explictly presents syntactic and semantic dependencies, such as anaphora relations, in a same graph.

In the FDG framework, several levels of analysis can be maintained as follows. Let p be a element, which has the syntax predicate $\delta(n_1, p, f) \wedge f \in F$, and G another predicate where it holds $G \cap F = \emptyset$ and $\delta(n_2, p, g) \wedge g \in G$. Then, there can be a head nucleus n_2 that is not necessarily the same as the head nucleus n_1 in the syntax predicate. So far, we have not studied the properties of this kind of forest but it is a way to extend the analysis to cover multiple levels in natural language analysis.

Let us have an artificial language with three letters *a*, *b* and *c*. Also let there be a rule stating that there is the dependency predicate *ab-dep* between a preceding letter *a* and letter *b*. One or more of the instances of the letter *c* can appear between the letters *a* and *b*. This linear contextual restriction can be declared with a schematic definition like

$$a \Rightarrow \_\, c^* b,$$

where the underscore denotes the place of the letter on the left side of the arrow.

When we define the dependency predicate, we use nuclei as letters and the properties of the nuclei can be tested when needed. For instance, the previous definition could be refined as an expression

$$a \in X \text{ AND } b \in Z \text{ AND } c \in Y \text{ AND } X \Rightarrow \_\, Y^* Z,$$

where the properties of the nuclei are denoted with the expressions like $a \in X$, which can be interpreted here to mean that the token *a* belongs to the nucleus *X*, which is located in the place denoted by the arrow definition. Here, we can test properties of the nucleus, such as word-class or shallow syntactic function.

The following rule states that there is a dependency between nuclei *X* and *Z*, where the nuclei contain tokens *a* and *b*, respectively. In addition, the nuclei have to appear in the given context.

$$\delta(X, Z, \text{ab-dep}) \quad \text{IF} \quad a \in X \text{ AND } b \in Z \text{ AND } c \in Y \text{ AND } X \Rightarrow \_\, Y^* Z$$

3.3 Non-generative approach

Neither the regular language module nor the dependency structure of FDG is generative in Chomsky's (1957) sense. Chomsky maintains that the grammar should first be able to generate all and only the "grammatical" sentences of English, or any other natural language. Thus the parser would be able to accept or reject sentences according to whether the input is a grammatical sentence or not. According to him (Chomsky, 1957, p. 23), regular languages are not very useful in analysing English syntactically because they can not produce all and only the grammatical sentences of the language.

FDG is a parsing framework which is not intended to maintain the grammaticality of an input sentence. Our aim is to parse also ungrammatical sentences. When using the regular language backbone, we have to design the regular language module so that it accepts all the legitimate analyses and rejects some illegitimate ones. Some ambiguity is left pending and the more powerful structural dependency analysis is applied to resolve this pending ambiguity. Furthermore, FDG builds dependency structures as long as this can be done by using the given declarations. The well-formedness, completeness or grammaticality of a sentence should be interpreted from the output afterwards, if needed.

3.4 Expression power

In this chapter, we briefly consider examples which show the dependency structures to be more powerful than regular languages. The first example shows how FDG can create dependencies emulating an unlimited amount of embeddings or bracketings. The other example shows a special case of

1	$\delta(X, Y, \text{pair})$	IF	"[" $\in X$ AND "]" $\in Y$
		AND	$Y \Rightarrow X\_$;
2	$\delta(X, Y, \text{inner})$	IF	"[" $\in X$ AND "[" $\in Y$
		AND	$Y \Rightarrow X\_$;
3	$\delta(X, Y, \text{parallel})$	IF	"[" $\in X$ AND "[" $\in Y$
		AND	$\exists Z$: "[" $\in Z : Y \Rightarrow Z\_$
		AND	$\delta(X, Z, \text{pair})$;
4	$\delta(X, Y, \text{pair})$	IF	"[" $\in X$ AND "]" $\in Y$
		AND	$\exists Z$: "]" $\in Z : Y \Rightarrow Z\_$
		AND	$\exists V$: "[" $\in V : \delta(V, Z, \text{pair})$
		AND	$\exists W$: "[" $\in W : \delta^*(W, V, \text{parallel})$
		AND	$\delta(X, W, \text{inner})$;

Figure 10: Bracketing in FDG

indexed languages, namely $a^n b^n c^n$, which does not belong to the context-free grammars.

3.4.1 Context-free languages

Theorem: The FDG formalism can attach dependencies between bracket pairs of [and] so that balanced matching pairs have mutual dependency. Proof: Let the pairing of the bracketing be the functional dependencies between the opening bracket and the closing bracket. Let the depth of the bracketing be the maximum amount pairings inside.

With a simple bracketing such as '[ ]', rule number 1 in Figure 10 is sufficient to create the pairing: let us take any token]. If the previous token is [then there is the dependency predicate *pair* whose head is the token [.

Let us suppose that the pairing works for the pairings of the depth $n-1$ and less. Let us take a pairing p of the depth n. The depths of all the pairings p_i inside the pairing p are equal or less than $n-1$. Therefore, through induction, each pairing p_i has the dependency *pair* with the head [.

Next, we introduce two auxiliary dependency types. The leftmost pairing p_1 of the pairings p_i has the dependency *inner* from the opening bracket of the pairing p_1 as a dependent to the opening bracket of pairing p. This is declared in rule number 2 in Figure 10: any sequence of opening brackets '[[' has the dependency *inner* between so that the previous bracket is the head. Therefore the pairing p_1 is a dependent of the pairing p.

Furthermore, all consequent pairings p_i have the dependency *parallel* between them so that the pairing p_i is the head and the pairing p_{i+1} is the

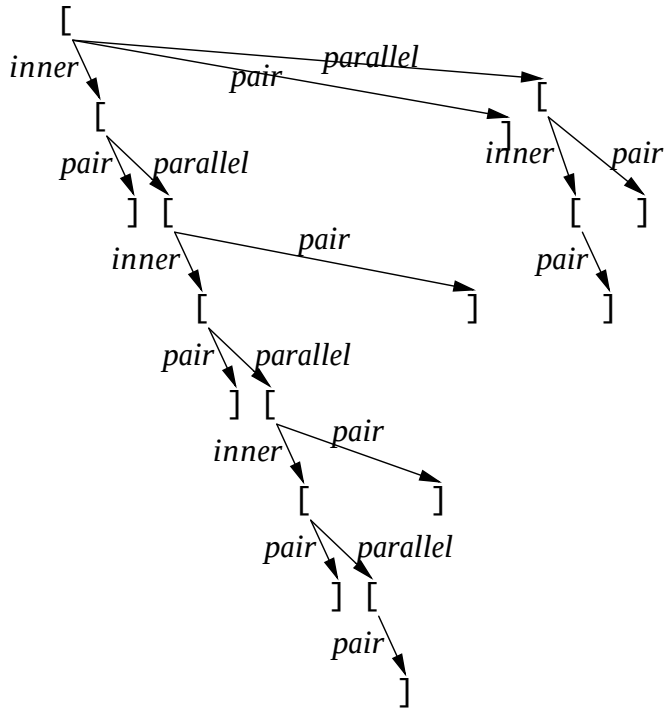

Figure 11: Visualised context-free bracketing

dependent. This is declared in rule number 3 in Figure 10: any sequence of a pairing and an opening bracket '[...] [' has a dependency *parallel*, so that the head is the first opening bracket which has the dependency *pair*, whose dependent is the right bracket] that is, the previous token before the second opening bracket.

Finally, the pairing p of the depth n can be done because the following conditions hold: the closing bracket of the pairing p is preceded by the closing bracket c_n of the pairing p_n. The opening bracket o_n is the head of the dependency pair, which exists because the depth of the pairing p_n is $n-1$ or less. There are one or more pairings p_i. All of them have the dependency *parallel* between the pairings p_i and p_{i+1}. The opening bracket o_1 of the pairing p_1 is the uppermost head of this chaining. The opening bracket before the bracket o_1 is the opening bracket of the pairing p. This is declared in rule number 4 in Figure 10.

The dependency structure built in the previous proof is visualised with the brackets `[[][[][[][]]]][[]]`. The output of FDG using the mini-

5	$\delta(X, Y, \text{pair})$	IF	"]" $\in X$ AND ">" $\in Y$
		AND	$Y \Rightarrow X\_$;
6	$\delta(X, Y, \text{pair})$	IF	"]" $\in X$ AND ">" $\in Y$
		AND	$\exists Z =$ ">" : $Y \Rightarrow Z\_$
		AND	$\exists U$: "]" $\in U : \delta(U, Z, \text{pair})$
		AND	$\exists V$: "[" $\in V : \delta(V, U, \text{pair})$
		AND	$\exists W$: "[" $\in W : \delta(V, W, \text{inner})$
		AND	$\delta(W, X, \text{pair})$

Figure 12: Additional rules for non-contiguous bracketing $[^n\,]^n{>}^n$

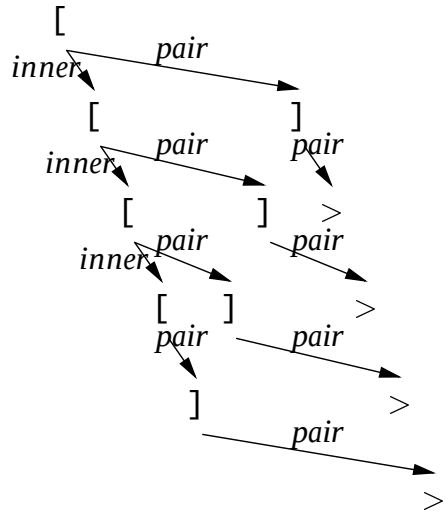

Figure 13: Visualised extended bracketing

grammar in Figure 10 is in Figure 11.

3.4.2 Indexed languages

Theorem: The FDG formalism can attach dependencies for the sequence $a^n b^n c^n = a_1 a_2 \ldots a_n b_1 b_2 \ldots b_n c_1 c_2 \ldots c_n$, creating a dependency chain for token sequences: $a_1 b_n c_1$, $a_2 b_{n-1} c_2$, $\ldots a_n b_1 c_n$.
Proof: Let us select $a =$ [and $b =$]. The mini-grammar in Figure 10 declares pairings $a_1 b_n$, $a_2 b_{n-1}$, $\ldots a_n b_1$. Now, let us select $c =$ >. We can then create the triple $a_1 b_n c_1$ by using rule number 5 in Figure 12. Any token > is a dependent of the preceding token], although in the language

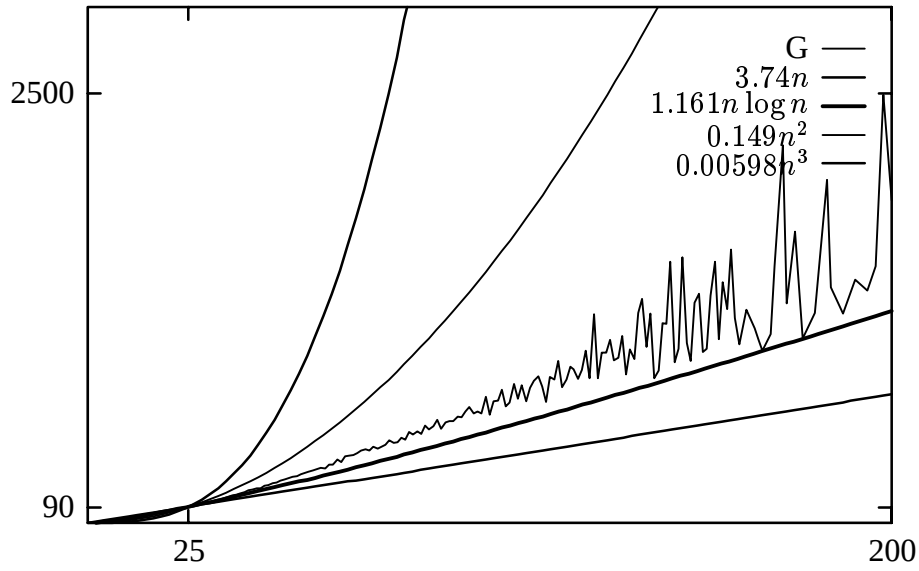

Figure 14: Average running time of FDG in Gutenberg texts

$a^n b^n c^n$ there is only one such token.

By using induction, we first assume the triple

$$a_{i-1} b_{n-(i-1)-1} c_{i-1} = a_{i-1} b_{n-i} c_{i-1}$$

is declared correctly. Let us start from the token c_i. The previous token c_{i-1} belongs to the triple $a_{i-1} b_{n-i} c_{i-1}$, where a_{i-1} can be found by following two times the dependency *pair* towards the head. The mini-grammar in Figure 10 declares that the token a_{i-1} is the head of the token a_i, whereas the token a_i is the head of the token b_{n-i+1}. This token is the desired head of the token c_i. This is declared in rule number 6 in Figure 12.

The dependency structure built in the previous proof is visualised with the brackets [[[[]]]]>>>>. The output of FDG using the mini-grammar in Figures 10 and 12 is in Figure 13.

3.5 Average running time of FDG

The average running time is tested using the same method as used in testing CG-2 earlier in this summary. The linguistic description by Järvinen (1998) has non-contiguous structures which can not be described in context-free languages, but the degree of the complexity is hard to approximate. FDG is applied to the same material as CG-2 above. The result is

in Figure 14 where the running time curve roughly follows the $O(n \log n)$ curve. One can not make clear conclusions from these few tests on the running time because they obviously depend on the linguistic description and the properties of the given language. Nevertheless, this shows that there exists a broad-coverage grammar for English which runs fast.

4 Conclusion

A group of reductionist parsing formalisms based on disambiguation in terms of regular languages are presented. The novel parsing framework, *Functional Dependency Grammar*, combines such reductionist formalisms with explicit dependency structure. The dependency graphs, however, are based on the relations between the elements of the sentence.

I have presented a complete system for natural language processing from tokenisation to surface syntax. The non-determistic tokenisation method splits a sentence into several, possibly ambiguous, tokens. The tokenisation is combined with multiword unit recognition and lexical analysis. A token is the basic element for the reductionist surface-syntactic parsers and a nucleus, which is a contiguous or non-contiguous sequence of tokens, is the basic element of the dependency structures.

The Functional Dependency Grammar parser has expression power which exceeds context-free grammars. It applies regular language components using finite-state constraint languages and depedency graphs, which are closely related to Tesnière's Structural Syntax (dependency grammar). This means that the syntactic element is not a word but a nucleus which can be a non-contiguous sequence of tokens that are defined as words or parts of words. The dependencies between nuclei form a tree.

The application of FDG to English by Timo Järvinen has been tested. It was shown that the approach can be used for describing a broad-coverage grammar for natural language. Furthermore, the practical parser produces accurate results and runs fast.

References

Abney, Steven P. 1991. Parsing by chunks. In Robert C. Berwick, Steven P. Abney, and Carol Tenny, editors, *Principle-Based Parsing: Computation and Psycholinguistics*, volume 44 of *Studies in Linguistics and Philosophy*. Kluwer Academic Publishers, Dordrecht / Boston / London, pages 257–278.

Aduriz, Itziar, Izaskun Aldezabal, Iñaki Alegria, Xabier Artola, Nerea Ezeiza, and Ruben Urizar. 1996. Euslem: A lemmatiser / tagger for Basque. In Martin Gellerstam, Jerker Järborg, Sven-Göran Malmgren, Kerstin Norén, Lena Rogström, and Catarina Röjder Papmehl, editors, *Papers submitted to the Seventh EURALEX International Congress on Lexicography*, pages 17–26, Göteborg, Sweden. Göteborg University, Department of Swedish.

Aït-Mokhtar, Salait and Jean-Pierre Chanod. 1997. Incremental finite-state parsing. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, pages 72–79, Washington, D.C., April. Association for Computational Linguistics.

Anttila, Arto. 1995. How to recognise subjects in English. In Karlsson et al. (1995), chapter 9, pages 315–358.

Bick, Eckhard. 1997. Dependensstrukturer i constraint grammar syntaks for portugisisk. In Tom Brøndsted and Inger Lytje, editors, *Sprog og Multimedier*, Denmark, Aalborg. Aalborg Universitetsforlag.

Brants, Thorsten and Christer Samuelsson. 1995. Tagging the teleman corpus. In Kimmo Koskenniemi, editor, *Proceedings of the 10th Nordic Conference of Computational Linguistics*, Publications 26, Department of General Linguistics, pages 7–20, Helsinki, Finland, May. University of Helsinki, Yliopistopaino.

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

Chanod, Jean-Pierre and Pasi Tapanainen. 1995. Tagging French – comparing a statistical and a constraint-based method. In *Proceedings of the Seventh Conference of the European Chapter of the Association for Computational Linguistics*, pages 149–156, Dublin, Ireland, March. Association for Computational Linguistics.

Chanod, Jean-Pierre and Pasi Tapanainen. 1996a. A non-deterministic tokeniser for finite-state parsing. In A. Kornai, editor, *Extended finite state models of language*, pages 10–11, Budapest, Hungary. ECAI'96.

Chanod, Jean-Pierre and Pasi Tapanainen. 1996b. A robust finite-state parser for French. In John Carroll, editor, *Workshop on Robust Parsing*, pages 16–25, Prague, Czech, August. Eight European Summer School In Logic, Language and Information.

Chanod, Jean-Pierre and Pasi Tapanainen. 1999. Finite state based reductionist parsing for French. In Kornai (1999), chapter 8, pages 72–85.

Chomsky, Noam. 1957. *Syntactic Structures*. Number 4 in Janua Linguarum. Mouton & Co., The Hague, Paris.

Collins, Michael John. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, USA, June. Association for Computational Linguistics.

Eisner, Jason M. 1996. Three new probabilistic models for dependency parsing: An exploration. In *The 16th International Conference on Computational Linguistics*, volume 1 of *COLING-96*, pages 340–345, Copenhagen, Denmark, August. Center for Sprogteknologi, COLING-96 Organizing Committee.

Engel, Ulrich. 1996. Tesnière mißverstanden. In Gertrud Gréciano and Helmut Schumacher, editors, *Lucien Tesnière – Syntaxe structurale et operations mentales. Akten des deutsch-französischen Kolloquiums anläßlich der 100. Wiederkehr seines Geburtstages, Strasbourg 1993*, volume 348 of *Linguistische Arbeiten*. Niemeyer, Tübingen, pages 53–61.

Francis, W. Nelson and Henry Kučera. 1982. *Frequency Analysis of English Usage: Lexicon and Grammar*. Houghton-Mifflin Company, Boston.

Fraser, Norman M. 1989. Parsing and dependency grammar. *UCL working papers in Linguistics*, 1:296–319.

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

Giguet, Emmanuel and Jacques Vergne. 1997. Syntactic structures of sentences from large corpora. In *Fifth Conference on Applied Natural Language Processing – Descriptions of System Demonstrations and Videos*,

pages 1–2, Washington, D.C., March. Association for Computational Linguistics.

Greene, Barbara B. and Gerald M. Rubin. 1971. Automatic grammatical tagging of English. Technical report, Department of Linguistics, Brown University, Providence, Rhode Island.

Grefenstette, Gregory and Pasi Tapanainen. 1994. What is a word, what is a sentence? problems of tokenization. In *The 3rd International Conference on Computational Lexicography*, pages 79–87, Budapest. Research Institute for Linguistics Hungarian Academy of Sciences.

Hajič, Jan. 1998. Building a syntactically annotated corpus: The prague dependency treebank. In Eva Hajičová, editor, *Issues of Valency and Meaning – Studies in Honour of Jarmila Panevová*. Karolinum – Charles University Press, Prague, pages 106–132.

Hajičová, Eva. 1998. Movement rules revisited. In Kahane and Polguère (1998), pages 49–57.

Harris, Zellig. 1962. *String Analysis of Sentence Structure*. Mouton & Co., The Hague, Paris.

Hays, David G. 1964. Dependency theory: A formalism and some observations. *Language*, 40(4):511–525.

Heringer, Hans Jürgen. 1993. Dependency syntax – formalized models. In Joachim Jacobs, Arnim von Stechow, Wolfgang Sternefeld, and Theo Vennemann, editors, *Syntax – An International Handbook of Contemporary Research*, volume 1. Walter de Gruyter, Berlin - New York, chapter 13, pages 316–328.

Holan, Tomáš, Vladislav Kuboň, and Martin Plátek. 1997. A prototype of a grammar checker for Czech. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, pages 147–154, Washington, D.C., April. Association for Computational Linguistics.

Hudson, Richard. 1984. *Word Grammar*. Basil Blackwell, Oxford.

Hudson, Richard. 1991. *English Word Grammar*. Basil Blackwell, Cambridge, MA.

Hurskainen, Arvi. 1996. Disambiguation of morphological analysis in Bantu languages. In *The 16th International Conference on Computational Linguistics*, volume 1 of *COLING-96*, pages 568–573, Copenhagen,

Denmark, August. Center for Sprogteknologi, COLING-96 Organizing Committee.

Joshi, Aravind K. and Philip Hopely. 1999. A parser from antiguity: an early application of finite state transducers to natural language parsing. In Kornai (1999), chapter 2, pages 6–15.

Järvinen, Timo. 1994. Annotating 200 million words: The bank of English project. In *The 15th International Conference on Computational Linguistics Proceedings*, volume I of *COLING 94*, pages 565–568, Kyoto, Japan, August. ICCL, COLING 94 Organizing Committee.

Järvinen, Timo. 1998. Tesnière's structural syntax reworked. Master's thesis, Department of General Linguistics, University of Helsinki, Finland, December.

Järvinen, Timo and Pasi Tapanainen. 1997. A dependency parser for English. Technical Report TR-1, Department of General Linguistics, University of Helsinki, Finland, February.

Järvinen, Timo and Pasi Tapanainen. 1998. Towards an implementable dependency grammar. In Kahane and Polguère (1998), pages 1–10.

Kahane, Sylvain and Alain Polguère, editors. 1998. *Processing of Dependency-Based Grammars*, COLING-ACL'98, Montreal, Canada, August. Association for Computational Linguistics, Universitè de Montréal.

Karlsson, Fred. 1990. Constraint grammar as a framework for parsing running text. In Hans Karlgren, editor, *Papers presented to the 13th International Conference on Computational Linguistics*, volume 3, pages 168–173, Helsinki, Finland, August. ICCL, Yliopistopaino, Helsinki.

Karlsson, Fred. 1995. The formalism and environment of constraint grammar parsing. In Karlsson et al. (1995), chapter 2, pages 41–88.

Karlsson, Fred, Atro Voutilainen, Juha Heikkilä, and Arto Anttila, editors. 1995. *Constraint Grammar : a language-independent system for parsing unrestricted text*, volume 4 of *Natural Language Processing*. Mouton de Gruyter, Berlin and New York.

Karttunen, L., J-P. Chanod, G. Grefenstette, and A. Schiller. 1997. Regular expressions for language engineering. *Natural Language Engineering*, 2(4):307–330.

Karttunen, Lauri. 1994. Constructing lexical transducers. In *The 15th International Conference on Computational Linguistics Proceedings*, volume I of *COLING 94*, pages 406–411, Kyoto, Japan, August. ICCL, COLING 94 Organizing Committee.

Karttunen, Lauri, Ron Kaplan, and Annie Zaenen. 1992. Two-level morphology with composition. In *Proceedings of the Fourteenth International Conference on Computational Linguistics*, volume I of *COLING-92*, pages 141–148, Nantes, France, August. ICCL.

Karttunen, Lauri and Kemal Oflazer, editors. 1998. *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, Ankara, Turkey, June. Department of Computer Engineering and Information Science, Bilkent University, Bilkent University.

Kornai, András, editor. 1999. *Extended Finite State Models of Language*. Studies in Natural Language Processing. Cambridge University Press.

Koskenniemi, Kimmo. 1983. *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production*. Publications 11, Department of General Linguistics, University of Helsinki. Yliopistopaino, Helsinki.

Koskenniemi, Kimmo. 1990. Finite-state parsing and disambiguation. In Hans Karlgren, editor, *Papers presented to the 13th International Conference on Computational Linguistics*, volume 2, pages 229–232, Helsinki, Finland, August. ICCL, Yliopistopaino, Helsinki.

Koskenniemi, Kimmo. 1997. Representations and finite-state components in natural language. In Roche and Schabes (1997), chapter 3, pages 99–116.

Koskenniemi, Kimmo, Pasi Tapanainen, and Atro Voutilainen. 1992. Compiling and using finite-state syntactic rules. In *Proceedings of the Fourteenth International Conference on Computational Linguistics*, volume I of *COLING-92*, pages 156–162, Nantes, France, August. ICCL.

Lepage, Yves, Ando Shin-Ichi, Akamine Susumu, and Iida Hitoshi. 1998. An annotated corpus in Japanese using Tesnière's structural syntax. In Kahane and Polguère (1998), pages 109–115.

Lin, Dekang. 1996. Evaluation of Principar with the Susanne corpus. In John Carroll, editor, *Workshop on Robust Parsing*, pages 54–69, Prague,

Czech, August. Eight European Summer School In Logic, Language and Information.

Lindberg, Nikolaj and Martin Eineborg. 1998. Learning constraint grammar-style disambiguation rules using inductive logic programming. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume II of *COLING-ACL'98*, pages 775–779, Montreal, Canada, August. ICCL, Association for Computational Linguistics.

McCord, Michael. 1990. Slot grammar: A system for simpler construction of practical natural language grammars. In R. Studer, editor, *Natural Language and Logic: International Scientific Symposium*, Lecture Notes in Computer Science, pages 118–145. Springer, Berlin.

Mel'čuk, Igor A. 1987. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany.

Oflazer, Kemal and İlker Kuruöz. 1994. Tagging and morphological disambiguation of Turkish text. In *Proceedings of the 4th Conference on Applied Natural Language Processing*, pages 144–149, Stuttgart, Germany, October. Association for Computational Linguistics.

Palmer, David D. and Marti A. Hearst. 1994. Adaptive sentence boundary disambiguation. In *Proceedings of the 4th Conference on Applied Natural Language Processing*, pages 78–83, Stuttgart, Germany, October. Association for Computational Linguistics.

Piitulainen, Jussi. 1995. Locally tree-shaped sentence automata and resolution of ambiguity. In Kimmo Koskenniemi, editor, *Proceedings of the 10th Nordic Conference of Computational Linguistics*, Publications 26, Department of General Linguistics, pages 50–58, Helsinki, Finland, May. University of Helsinki, Yliopistopaino.

Robinson, Jane J. 1970. Dependency structures and transformational rules. *Language*, 46:259–285.

Roche, Emmanuel and Yves Schabes, editors. 1997. *Finite-state language processing*. A Bradford Book. MIT Press, Cambridge, Massachusetts.

Samuelsson, Christer, Pasi Tapanainen, and Atro Voutilainen. 1996. Inducing constraint grammars. In Laurent Miclet and Colin de la Higuera, editors, *Grammatical Inference: Learning Syntax from Sentences*,

volume 1147 of *Lecture Notes in Artificial Intelligence*, pages 146–155, Montpellier, France, September. ICGI'96, Springer.

Samuelsson, Christer and Atro Voutilainen. 1997. Comparing a linguistic and a stochastic tagger. In *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 246–253, Madrid, Spain, July. Association for Computational Linguistics.

Segond, Frédérique and Pasi Tapanainen. 1995. Using a finite-state based formalism to identify and generate multiword expressions. Technical Report MLTT-019, Rank Xerox Research Centre, Grenoble Laboratory, France.

Sgall, Petr, Eva Hajičová, and Jarmila Panevová. 1986. *The Meaning of the Sentence in its Semantic and Pragmatic Aspects*. D. Reidel, Dordrecht.

Skut, Wojciech, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. Software for annotating argument structure. In *Fifth Conference on Applied Natural Language Processing – Descriptions of System Demonstrations and Videos*, pages 27–28, Washington, D.C., March. Association for Computational Linguistics.

Sleator, Daniel and Davy Temperley. 1991. Parsing English with a link grammar. Technical Report CMU-CS-91-196, Carnegie Mellon University, October.

Strzalkowski, Tomek, Gees Stein, G. Bowden Wise, Jose Perez-Carballo, Pasi Tapanainen, Timo Järvinen, Atro Voutilainen, and Jussi Karlgren. 1999. Natural language information retrieval: TREC-7 report. In Voorhees and Harman (1999), pages 217–226.

Tapanainen, Pasi. 1991. Äärellisinä automaatteina esitettyjen kielioppisääntöjen soveltaminen luonnollisen kielen jäsentässä. Master's thesis, Department of Computer Science, University of Helsinki, Finland, September.

Tapanainen, Pasi. 1992. *Äärellisiin automaatteihin perustuva luonnollisen kielen jäsennin*. Licentiate thesis, Department of Computer Science, University of Helsinki, Finland.

Tapanainen, Pasi. 1995. RXRC finite-state compiler. Technical Report MLTT-020, Rank Xerox Research Centre, Grenoble Laboratory, France.

Tapanainen, Pasi. 1996. *The Constraint Grammar Parser CG-2*. Publications 27, Department of General Linguistics, University of Helsinki. Yliopistopaino, Helsinki.

Tapanainen, Pasi. 1997. Applying a finite-state intersection grammar. In Roche and Schabes (1997), chapter 10, pages 311–327.

Tapanainen, Pasi and Timo Järvinen. 1994. Syntactic analysis of natural language using linguistic rules and corpus-based patterns. In *The 15th International Conference on Computational Linguistics Proceedings*, volume I of *COLING 94*, pages 629–634, Kyoto, Japan, August. ICCL, COLING 94 Organizing Committee.

Tapanainen, Pasi and Timo Järvinen. 1997. A non-projective dependency parser. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, pages 64–71, Washington, D.C., April. Association for Computational Linguistics.

Tapanainen, Pasi and Timo Järvinen. 1998. Dependency concordances. *International Journal of Lexicography*, 11(3):187–203, September.

Tapanainen, Pasi, Jussi Piitulainen, and Timo Järvinen. 1998. Idiomatic object usage and support verbs. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume II of *COLING-ACL'98*, pages 1289–1293, Montreal, Canada, August. ICCL, Association for Computational Linguistics.

Tapanainen, Pasi and Atro Voutilainen. 1993. Ambiguity resolution in a reductionistic parser. In *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics*, page 475, Utrecht, the Netherlands, April. EACL, Association for Computational Linguistics.

Tapanainen, Pasi and Atro Voutilainen. 1994. Tagging accurately – don't guess if you know. In *Proceedings of the 4th Conference on Applied Natural Language Processing*, pages 47–52, Stuttgart, Germany, October. Association for Computational Linguistics.

Tesnière, Lucien. 1959. *Éléments de syntaxe structurale*. Librairie Klincksieck, Paris.

Voorhees, E. M. and D.K. Harman, editors. 1999. *The Seventh Text Retrieval Conference (TREC-7)*, NIST Special Publications 500-242, Gaithersburg, Maryland. National Institute of Standards and Technology.

Voutilainen, Atro. 1994. *Three studies of grammar-based surface parsing of unrestricted English text*. Ph.D. thesis, Publications 24, Department of General Linguistics, University of Helsinki, Finland, March.

Voutilainen, Atro. 1995. Morphological disambiguation. In Karlsson et al. (1995), chapter 6, pages 165–284.

Voutilainen, Atro. 1997. Designing a (finite-state) parsing grammar. In Roche and Schabes (1997), chapter 9, pages 283–310.

Voutilainen, Atro. 1998. Does tagging help parsing? A case study on finite state parsing. In Karttunen and Oflazer (1998), pages 25–36.

Voutilainen, Atro, Juha Heikkilä, and Arto Anttila. 1992. *Constraint Grammar of English – A Performance-Oriented Introduction*. Publications 21, Department of General Linguistics, University of Helsinki. Yliopistopaino, Helsinki.

Voutilainen, Atro and Pasi Tapanainen. 1993. Ambiguity resolution in a reductionistic parser. In *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics*, pages 394–403, Utrecht, the Netherlands, April. EACL, Association for Computational Linguistics.

Yli-Jyrä, Anssi. 1995. Schematic finite-state intersection parsing. In *Short Papers Presented at the 10th Nordic Conference of Computational Linguistics*, pages 95–103, Helsinki, Finland, May.