AUTOMATIC FUNCTOR ASSIGNMENT IN THE PRAGUE DEPENDENCY TREEBANK

A step towards capturing natural language semantics

Zdeněk Žabokrtský

ÚFAL/CKL Technical Report TR–2001–10

March 2001

Abstract

The goal of this thesis is to design, implement and evaluate a software tool that should reduce the huge amount of human work involved in the development of the Prague Dependency Treebank. The PDT is a research project at the Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University, Prague. It is aimed at a complex annotation of a part of the Czech National Corpus, built at the Institute of the Czech National Corpus, Faculty of Philosophy, Charles University. The annotation scheme comprises three levels: morphological, analytical, and tectogrammatical. At the last level, each autosemantic word of a sentence is annotated with its tectogrammatical function (functor) that represents its linguistic meaning within the sentence, e.g., Actor, Patient, Addressee, various types of spatial and temporal circumstantials, Means, Manner, Extent, Consequence, Condition. Manual annotation of functors naturally is very time-consuming. The motivation for this thesis is the fact that a system for Automatic Functor Assignment (AFA) (i.e., a system which could automatically assign at least some of the functors), would save the time of the experts and possibly accelerate the growth of the PDT.

For the purposes of development, the data, which were already manually annotated, were split into training and testing sets. After observing various characteristics of this data, I proposed and implemented four complementary families of methods of the AFA: methods based on handwritten rules, methods based on automatically extracted dictionaries, a method based on the notion of nearest vector in feature space, and a method based on Machine Learning. The training set played a crucial role for the development of the last three of them. Besides the implemented methods, I outline several alternative approaches to the AFA.

The implementation of the presented AFA system consists of many small programs for data preprocessing, functor assigning, and performance evaluation. It was implemented in the Linux environment. Most of the code was written in Perl. All the programs are applied on the data in a strictly pipeline fashion. In this way, the whole system remains open for further extensions.

The implementation was tested on the testing set. The performance (cover, precision, etc.) of individual functor-assigning components was measured and evaluated in detail.

Abstrakt

Cílem této práce bylo navrhnout, implementovat a vyhodnotit softwarový nástroj, který by pomohl snížit obrovské množství lidské práce potřebné pro vytvoření Pražského závislostního korpusu. Pražský závislostní korpus je výzkumný projekt realizovaný v Institutu formální a aplikované lingvistiky při Matematicko-fyzikální fakultě Univerzity Karlovy v Praze. Cílem tohoto projektu je komplexní anotace části Českého národního korpusu. Anotační schéma zahrnuje tři úrovně: morfologickou, analytickou a tektogramatickou. Na poslední z nich je každému autosémantickému slovu přiřazena jeho tektogramatická funkce (funktor), která zachycuje jeho význam ve větě, jako např. aktor, pacient, adresát, různé druhy časových a místních doplnění, prostředek, způsob, míra, důsledek, podmínka. Ruční anotace funktorů je přirozeně velmi náročná na čas vyškolených odborníků. Motivací pro tuto diplomovou práci tedy byla skutečnost, že jakýkoli systém automatického doplňování funktorů (Automatic Functor Assignment, AFA), který by dokázal přiřadit alespoň část funktorů, by snížil zátěž těchto odborníků a přispěl by tak k urychlení růstu Pražského závislostního korpusu.

Pro vývoj systému AFA jsem použil data, která už byla ručně anotovaná na tektogramatické rovině. Rozdělil jsem je na trénovací a testovací množinu. Navrhl jsem a implementoval čtyři vzájemně se doplňující skupiny metod automatické anotace: metody založené na ručně psaných pravidlech, metody založené na automaticky extrahovaných slovnících, metodu založenou na principu nejbližšího vektoru v příznakovém prostoru a metodu založenou na strojovém učení. Trénovací množina hrála klíčovou roli zejména pro poslední tři skupiny. Kromě metod, které jsem implementoval, uvádím ještě několik alternativních přístupů.

Výsledný systém AFA je realizován jako skupina několika menších programů pro předzpracování dat, pro přiřazování funktorů a pro vyhodnocení správnosti výsledků. Systém byl implementován pod operačním systémem Linux, většina kódu byla napsána v jazyce Perl. Veškeré zpracování dat, tj. předzpracování, přiřazení a vyhodnocení, je důsledně proudové ('roury'). Díky této koncepci může být systém v budoucnosti snadno rozšířen.

Funkčnost implementace byla ověřena na testovacích datech. Charakteristické vlastnosti jednotlivých metod (pokrytí, úspěšnost atd.) pro přiřazování funktorů byly naměřeny a jsou podrobně popsány.

Acknowledgements

I do not exactly recollect what was the very first stimulus for my personal "Linguistic Turn". But I do recollect my trembling knees two years ago when I first came to the Institute of Formal and Applied Linguistics. Having met my present supervisor Ivana Kruijff-Korbayová there, I was really very lucky. During the whole time (though our mutual distance during the last year usually varied between one and two thousand kilometers), Ivana provided me with study materials and contacts to other people, helped me a lot with my awkward English and, above all, never stopped encouraging me. I am aware of the fact that this thesis would never have come to existence without her help.

I am grateful also to other people from IFAL, especially to the head of the department Eva Hajičová for supporting the presentation of my work at the conference on Text, Speech and Dialogue 2000, to Jarmila Panevová, Alla Bémová, and Alena Böhmová for consultations about tectogrammatical functions, to Petr Pajas for writing a Perl interface between my modules and the Prague Dependency Treebank data, and to Geert-Jan M. Kruijff for many useful comments to the text of this thesis.

I am much obliged to Mirko Navara and Olga Štěpánková and to the CEEPUS and SOCRATES projects for enabling my study stays abroad. I spent six weeks at the Fuzzy Logic Laboratory Linz (Austria) that is led by Erich P. Klement; I worked one month at Jožef Stefan Institute in Ljubljana (Slovenia) under the supervision of Nada Lavrač and Tomaž Erjavec; I studied three months at the Department of Mathematics, University of Patras (Greece), being supervised by Costas Drossos and Panagis Karazeris. Each of these places somehow contributed to the final version of the present thesis. In particular, I would like to thank Sašo Džeroski from Jožef Stefan Institute for his help with applying a machine learning approach.

I am also grateful to my ex-roommates Rosťa Horčík, Martin Kovář, Zdenda

Pohl, Pavel Puta and Lukáš Trejtnar, my jolly companions during “the hell at FEL”, not only for the numerous wanderings through the mountains of Bohemia and Slovakia but also for the shared interest in careful observation of Homer Simpson’s life.

I have been very lucky. I appreciate all the nice and bright people whom I have met in the years of my study, all the marvellous places I have seen during my travels and all the splendid music I have heard, for now I can adore the beauty of the world more than before.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Die Grenzen meiner Sprache
bedeuten die Grenzen meiner Welt.

Ludwig Wittgenstein

The motivation for exploring natural language can be formulated in a variety of ways, depending on the audience. So let me start with the motivation that could attract a computer scientist.

The immense amount of data available on the World Wide Web undoubtedly exceeds that of any information source accessible to an individual within the history of mankind, and moreover is still rapidly growing. For a human, this fact unfortunately does not generally entail a "better knowledge" (in the sense of [DePryck–93]) about the world, since the information is scattered, imperfect (incomplete, inconsistent), redundant (this also contributes to an overload of a human perception, though the redundancy causes no troubles for a computer), and non-homogenous. Besides searching and visualizing the documents, the contemporary computer technology—as the culprit of this information overload—cannot help much, thus leaving us often confused and unsatisfied in the web labyrinth. Any development of "document processing technology" that goes beyond the text as a sequence of characters and that is related to its meaning, sense and content, is nowadays either accompanied with extreme difficulties (e.g., machine translation), or still remains outside the realm of automation.

Obviously, many difficulties, which arise during the development of software for more sophisticated and more fruitful processing of this incredibly unordered heap of data, are caused by the fact that most of the information on the Internet is expressed in natural language (of course, not only on the Internet; let us cite from [Čermák–99]: "Most of information about anything is to be found in language;

there are, in fact, very few areas of human life based to a higher degree on non-verbal symbols."). The perspective of having technology that "understands" (i.e., can work with the meaning of) natural language, at least to some limited extent, is then more than a sufficient motivation for computer science to cooperate with linguistics.

We can look at natural language also from the viewpoint of artificial intelligence. For example, Turing's well-known and broadly discussed imitation test of "thinking machines" implicitly presumes a possibility of a man-machine communication in natural language; he mentioned even questions concerning poetry. Therefore, if there is a way to create whatever we could call not only *artificial* but also *intelligent* according to his definition, then it must contain "natural language technology" as one of its cornerstones.

I believe that the growing availability of sophisticated and richly annotated language data—especially those containing a semantic annotation—will be a milestone in AI, similarly as the data precisely measured and carefully collected by Tycho de Brahe played a key role for Johannes Kepler's discovery of the fundamental laws of astrophysics. And if not a milestone, then at least the next step towards the elusive horizon described by Allan Turing: *"One day ladies will take their computers for walks in the park and tell each other 'My little computer said such a funny thing this morning!' "*

One of the conditions for serious research in the domain of Natural Language Processing is the availability of language resources. This term stands for sets of language data and descriptions in machine processable form, used for building, evaluating or operating natural language and speech systems. In this thesis, I attempt to participate in the building of a specific language resource, namely the Prague Dependency Treebank.

1.1 Aim of the thesis

The Prague Dependency Treebank (PDT) is a research project aimed at a complex annotation of (i.e., the addition of selected linguistic information to) a part of the Czech National Corpus (electronic collection of Czech texts from selected sources).

The annotation scheme of the PDT was developed by the research team of the

Institute of Formal and Applied Linguistics[1], Faculty of Mathematics and Physics, Charles University, Prague, and consists of three layers of annotation: morphological, analytical and tectogrammatical. On the tectogrammatical level, annotated sentences are represented in the form of a specific kind of dependency tree, a so-called tectogrammatical tree structure (TGTS), where every autosemantic word has its own node ([BPS–99], [BH–99]).

Each node is annotated with its tectogrammatical function (functor) that represents its linguistic meaning within the sentence, e.g., actor, patient, addressee, predicate, different types of spatial and temporal circumstancials, means, manner, modality, extent, consequence, condition, aim, appurtenance, etc.

Most of the functors have to be assigned manually, word after word, sentence after sentence. The huge amount of labor involved in manual annotation (the PDT contains more than 26 thousand sentences) obviously slows down the growth of the PDT on the tectogrammatical level. Therefore, decreasing the amount of manual annotation has been the motivation for developing a more complex system for the *Automatic Functor Assignment* (AFA) described in this thesis.

1.2 Summary

In Chapter 2, I briefly summarize a few basic notions from the domains of Natural Language Processing, Corpus Linguistics and Machine Learning. They are not a standard part of a computer scientist's education, but they are indispensable for the work on the topic of this thesis.

Chapter 3 describes the Prague Dependency Treebank. The reader is given information about the source and the amount of the textual data involved. The annotation principles and the meaning of the annotation on all three levels are described in more detail, examples of tree structures are presented.

In Chapter 4, more careful formulation of the AFA task is given and the initial situation before starting the work on the AFA is described. The minimal amount of information that is sufficient for the correct functor assignment is discussed. Further, the data preprocessing is explained and the available training and testing material is mentioned.

In Chapter 5 all the methods incorporated into the AFA system are shown.

[1] http://ufal.mff.cuni.cz

Namely, the methods based on dictionaries, rules, nearest vector, and machine learning. Then I sketch several alternative approaches that have not been implemented yet, or have only been implemented partially so far.

Chapter 6 concerns the implementation details of the AFA realization. The description of how to extend the current AFA system is included.

In Chapter 7 measurements of the performance of the developed system are presented and evaluated.

Chapter 8 contains conclusions, a discussion of the obtained results and an outline of future improvements.

Chapter 2

Prerequisities

2.1 Natural Language Processing

The simplest way to elucidate what the Natural Language Processing (NLP) area currently covers is to enumerate several possible examples of NLP applications:[1]

- *Text databases and information extraction*: finding appropriate documents in response to user-queries from a database of texts.
- *Machine translation*: translating documents from one (natural) language into another with the help of a computer.
- *Text summarizing*: extracting the most important information from large texts.
- *Text editors*: a thesaurus or a system for correction of typing or grammatical errors are useful assistents during a text preparation.
- *Automatic documentation drafting*: automatic generation of texts from underlying content representation (possibly in multiple languages simultaneously), e.g. [KK–99].
- *Man-machine communication*: voice communication for control of a machine, automated customer service over the telephone etc.
- *Human-human communication*: computer aids for people with disabilities.

The development of applications like these profits from having collections of natural language data at their disposal both for research and testing.

The attractiveness of many branches of NLP significantly increases in the age of Internet. Most of the information accessible on the web consists of text in

[1] A more detailed description can be found, e.g., in [Allen–95] or [Strossa–99].

natural language (usually in English), but its enormous amount is far beyond the limits of the "text processing potential" of a human.

After the invasion of computers into every-day life, many NLP applications have become of practical importance. However, this should not overshadow NLP's position in the scientific world.

NLP is a markedly interdisciplinary domain. The core academic discipline focused on computer-based NLP is usually called *computational linguistics* (CL). Broadly speaking, the aim of CL is to develop computational models of natural language generation and understanding. But in order to build a computational model of language, several other disciplines need to cooperate. They are especially:

- "classical" linguistics, psycholinguistics, sociolinguistics, cognitive science
- philosophy
- mathematics
- computer science
- artificial intelligence

Natural language and its structures are usually viewed at several levels. In [Allen–95] the following levels of language description are distinguished:[2]

1. *Phonetic and phonological knowledge* concerns how words are related to the sounds that realize them.
2. *Morphological knowledge* concerns how words are constructed from more basic meaning units called morphemes.
3. *Syntactic knowledge* concerns how words can be put together to from correct sentences and determines what structural role each word plays.
4. *Semantic knowledge* concerns what words mean and how these meanings combine in sentences.
5. *Pragmatic knowledge* concerns how sentences are used in different situations and how use affects the interpretation of the sentence.
6. *Discourse knowledge* concerns how the preceding sentences affect the interpretation of the next sentence.

[2]These distinctions are a matter of continuing debate.

7. *World knowledge* includes general knowledge that the language users must have in order to maintain conversation.

2.2 Corpus Linguistics

When creating, justifying or falsifying their hypotheses, linguists work with different information sources: their intuition, introspection, experiments, observation, corpora. The term *corpus* stands (on the most general level) for a collection of records of authentic usages of natural language. It is the material baseline which serves for the linguistic analysis and description, both of the written and spoken language ([Šulc–99]).

It is natural to prefer to acquire information about language use directly from naturally occurring text instead of using introspection or intuition. Moreover, some new phenomena, which were not described nor observed yet, can be discerned during work with large corpora. Corpora serve as a material source not only for linguistics, but also for research areas dealing with human thinking or culture. Therefore, the impact of corpora on linguistics (and other sciences) is steadily growing. Or in the words of František Čermák([Čermák–99]):

> At the turn of the century, linguistics is more and more dependent on corpora; at the same time it is evident that corpora become a primary source of information.

On the other hand, it can be supposed that there are still some linquists who resist the "corpus challenge", and are therefore sometimes called "armchair linguists", thus being the opposition to corpus linguists. Fillmore's smart caricatures of both groups can be found in Appendix A, the "problem" has been discussed also in [Lager–95].

Not surprisingly, most corpora have been developed for English, e.g., the Brown Corpus, the British National Corpus, and the Penn Treebank. In order to get a feeling about the amount of text in contemporary corpora, let us mention that their size is measured in the order of hundreds of millions of words. Most European languages have some sort of a corpus already as well, even if just a small one.

Nowadays, due to the prevalence of electronic corpora, the term corpus is used nearly exclusively for an electronically stored and computer-readable text collection. Some more detailed requirements for the size or the representativeness can also be specified.

Corpora can be classified with respect to several criteria:

- corpora of one language versus *parallel* corpora, i.e., containing corresponding texts in more languages (e.g., [Erjavec-Ide–98]),
- *diachronous* corpora reflect a language in a longer epoch, *synchronous* corpora maintain only such a time period with no significant language changes,
- corpora of *spoken* or *written* language,
- corpora containing texts of particular genre(s).

The content of contemporary corpora is usually not only plain text; the text is enriched by *annotation.* Note that the goal of this thesis—automatic functor assignment—is nothing else than adding a specific type of annotation. Karel Pala defines annotating in [Pala–99] as follows:

> Annotating consist of adding selected linguistic information to an existing corpus of written or spoken language. Typically, this is done by some kind of coding being attached (semi)automatically or manually to the electronic representation of the text.

For different purposes there are different types of annotation, for instance:

- *morphological tagging* adds the part of speech specification (POS) and morphological categories (gender, number, case, tense ...),
- *parsing* adds syntactical tags that usually represent tree structures of sentences,
- tagging of *anaphoric relations*,
- *prosodic* tagging.

When annotating text in order to capture more complicated phenomena (e.g., annotation on the semantic level), the output is biased by the involved theory. This is also the case with the tectogrammatical annotation concerned in this thesis.

The next section will be devoted to Machine Learning, since it is frequently used for corpus annotation.

2.3 Machine Learning

Learning can be viewed as the acquisition of new knowledge, improving performance with practice, changing behaviour due to experience ([RL–95]). Mitchel's definition of *machine learning* is this: "a computer program learns if it improves its performance at some task through experience."

Machine learning (ML) can be used for classification and prediction tasks, planning, problem solving, knowledge discovery etc. Learning can be either *symbolic* or *sub-symbolic*. In the former case, the learned knowledge is represented in some formalism, e.g., decision trees. In the latter case, the derived knowledge does not have the form of symbolic descriptions that are easily understandable to humans, e.g., weight vectors in neural networks, binary chromosomes in genetic algorithms.

One of the ML strategies is *inductive concept learning* ([LD–94]). Inductive concept learning means deriving general classification rules (concept descriptions) from the descriptions of instances (positive examples) and non-instances (negative examples) of the concept to be learned, if it is the case of *single concept learning*. In the case of *multiple concept learning*, the concepts are usually named *classes*. Instead of having only positive and negative examples, the instances can be classified into more classes.

ML can be either supervised or unsupervised:

- in *supervised learning*, we have a training set of instances whose classification is known (the training set corresponds to the experience mentioned above),
- in *unsupervised learning*, the classification within the training set is unknown before learning.

During the ML process, we can profit from having *a priori* knowledge about the concepts which we have before the learning. This knowledge is called *background knowledge*.

ML can be either incremental or non-incremental:

- incremental learning can improve its performance step by step, as the training set grows,
- non-incremental learning learns from the whole training set at once; if some new examples come, the learning must start from beginning.

Quinlan's ML system C4.5 that will be employed in this thesis, is a member of the family of TDIDT learning systems (Top Down Induction of Decision Trees). The knowledge learned by these systems is represented in the form of decision trees.

Chapter 3

The Prague Dependency Treebank

The Prague Dependency Treebank, which has been inspired by the activities resulting in the Penn Treebank, is a research project aimed at a complex annotation of (a part of) the Czech National Corpus (CNC) ([BH–99]).

The Prague Dependency Treebank (PDT) is based on a scheme of annotation developed by the research team of the Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University, Prague. The annotation procedures are formulated with the aim to reduce the manual work of the annotators to a minimum, while adding to the raw text as reliable linguistic information as possible.

The PDT comprises three layers of annotation:

1. The morphemic layer with about 3000 morphemic tag values; a tag is assigned to each word form of a sentence in the corpus.
2. The analytic tree structures (ATSs) with every word form and punctuation mark explicitly represented as a node of a rooted tree, with no additional nodes added (except for the root of the tree of every sentence) and with the edges of the tree corresponding to (surface) dependency relations.
3. The tectogrammatical tree structures (TGTSs) corresponding to underlying sentence representations.

The tectogrammatical level annotation is based on the framework of Functional Generative Description (FGD) as developed within the Prague School of Linguistics by Petr Sgall and his collaborators since the beginning of the 1960's (e.g., [SHP–86]). The following section contains only a very rough sketch of some basic FGD notions, the reader can find a better explanation in [SHP–86], [Kruijff–98] and the literature quoted there.

3.1 Functional Generative Description

FGD is a stratificational approach to the systematic description of language, showing the main principles and properties of a language from the perspective of several sequentially related strata. A linguistic function at one stratum is realized by a form in the next lower stratum, in this order:

1. Deep structure, or tectogrammatical representation
2. Morphonemics
3. Phonemics
4. Phonetics

A speaker's utterance is then supposed to be generated as follows. The speaker has a deep structure of the information he/she wants to convey. Then, on the stratum of morphemics, the surface structure, conceived as a sequence of strings, is formed. Its elements are subsequently transformed to the phonemics and finally to the phonetics level.

In FGD, special attention is paid to the following features of natural language:

1. *Dependency relations* (they are discussed later in this chapter).
2. *Coordination and apposition*, which arise when two or more entities are viewed as a whole, are represented as one more complex structure. For example, in the sentence *"Děvenka Štěstí a Mládenec Žal stáli mi za zády ... "*, the subject consists of two parts that modify the verb *"stáli"* together.
3. *Contextual boundness and nonboundness* make distinction between what the speaker presents as recoverable from the preceding context, and what is new (modifying).
4. *Deep word order* represents the ordering of dependency relations within the tectogrammatical structures, closely related to contextual (non)boundness.
5. *Grammatical coreference*, e.g., the relation of a relative pronoun to an antecedent noun (*"Yesterday I saw a girl who played the violin.*).

```
tačí se trochu vybavit , <nanosit> kupu listí a sena - já ho
ie Každý mistr by se měl <honosit> nějakým rekordem či jedin
anční tísni by měly dítě <donosit> .  Bezvýhradná povinnost p
í hladovění bude schopna <donosit> plod .  Mimochodem i u sou
evítané těhotenství tzv.  <donosit> a dítěte se vzdát ve pros
mž sedíme , nepostavil .  <Vynosit> tuny kamení na zádech , t
byl v nebezpečí a naděje <donosit> dítě žádná .  Jeden večer
6 - Živit mateř .  mlékem <Nanosit> 57 - Ukončit létání 58 -
odstatně větší a může se <honosit> řadou úctyhodných přívlas
vy , v pokoji nekouřit , <nenosit> domů alkohol .  Dodržovat
ve městě , které se mělo <honosit> jen svým " dělnickým hnut
```

Figure 3.1: Response from the CNC for the querry .+nosit

3.2 The textual data provided by the Czech National Corpus

The Czech National Corpus, now containing more than a 100 million running words, is being built since 1994 at the Institute of the Czech National Corpus (ICNC) at Charles University in Prague, Czech Republic. The goal of the project is to create and continuously update a representative textual basis of several hundred million running words which would meet both the scientific and general cultural needs of its prospective users. The core of the system is, of course, its synchronic part consisting of contemporary texts: journalistic and technical texts since 1990, prose and poetry since 1960 ([Čermák–99]).

A sample of this corpus, about 20 million running words, is accessible on the Internet at URL `http://ucnk.ff.cuni.cz`. For example, Fig. 3.1 shows a part of the response obtained from this Internet interface to the query `.+nosit` (i.e., find the occurrences of the words with the suffix "nosit").

The CNC contains also morphosyntactic tagging, which is freely available for research purposes.

For the PDT purposes, a subset of the textual data was selected from the CNC as follows ([Hajič–98]):

- general newspaper articles, including but not limited to politics, sport, culture, hobby (newspapers Lidové noviny and Mladá fronta) – 40 %

- economic news and analysis Českomoravský profit – 20 %
- popular science magazine Vesmír – 20 %
- information technology texts – 20 %.

This sample contains alltogether 456 705 tokens (both words and punctuation marks) in 26610 sentences. This data was divided into 576 files, each containing up to 50 sentences.

3.3 Three levels of the PDT

The Prague Dependency Treebank has a three-level annotation structure. Full morphological tagging is available at the lowest level. The middle level provides syntactic annotation using dependency syntax; it is called the analytical level. The highest level of annotation is the tectogrammatical level, or the level of linguistic meaning [Hajič–98].

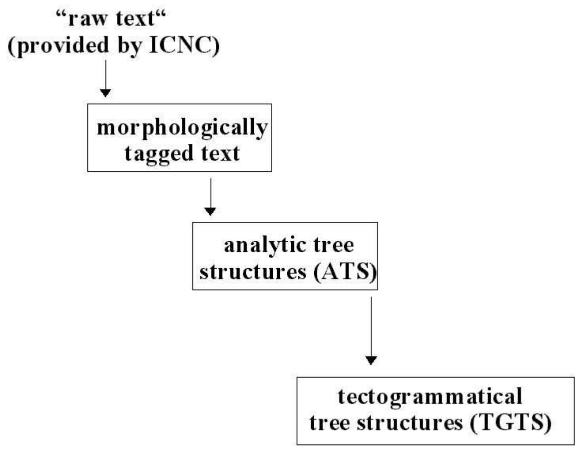


Figure 3.2: The layered structure of the PDT

3.3.1 Morphological annotation level

On the morphological level, a *morphological tag* and a *lemma* is assigned to each word form in the input text, the annotation contains no syntactic structure ([Hajič–98]).

I am going to describe the notation of the morphological tagging in detail here, because its understanding will be important later, namely for the discussion of data preprocessing (in particular, feature selection and extraction) in Section 4.5.

A morphological tag is a string consisting of two parts:

- prefix, part of speech (noun, adjective, pronoun, numeral, verb, adverb, preposition, conjunctions, particles), and possibly some more detailed specification (e.g., pronouns can be personal, reflexive, possessive, indefinite ...)
- suffix, specification of tag variables.

There are six tag variables corresponding to the following morphological categories:

1. *number* (abbreviation n), possible values are singular (in the morphological tag denoted as S), plural (P), dual (D), both, or special combination (X)
2. *case* (c), possible values are nominative (1), genitive (2), dative (3), accusative (4), vocative (5), locative (6), instrumental (7), underspecified value (X)
3. *gender* (g), possible values are masculine animate (M), masc. inanimate (I), feminine (F), neuter (N), any (X), masculine M or I (Y), not masculine (H), not masculine, but in special combinations only (Q), masc. inanimate or feminine (T), not feminine (Z), not masculine inanimate, not feminine (W)
4. *degree* of comparison (d), possible values are positive (1), comparative (2), superlative (3)
5. *person* (p, f), possible values are first (1), second (2), third (3), underspecified value (X)
6. *negation* (a), possible values are affirmative (A), negated form (N).

Examples of morphological tags:

- the tag for a verb in indicative mood and present tense is of the form VPnpa. For the word form *čteme* (we *read*) the tag variables (in this case only number, person, negation) are to be filled like this: VPP1A (i.e., plural, first person, affirmative),

- noun: Ngnca, *hrochovi* (to a *hippotamus*, singular number, dative case, animate gender) NMS3A
- reflexive possessive pronoun: PRSgnc
- ordinal numeral: CRgnc
- adverb: DB (there is no further specification of morphological categories, i.e., no variables)
- preposition: Rc

Czech is an inflectionally rich language (namely, there is a rich set of suffixes), therefore the full tag set contains currently as many as 3030 tags.

The Standard Generalized Markup Language (SGML) is used for the annotation on the morphological level. An example of a SGML tagged sentence is in Figure 3.3. Each contains one token (a word or a punctuation mark) from the annotated text. The element starting with the unpair tag `<MMl>` contains an automatically assigned lemma. The element starting with `MMt` contains the morphological tag.[1]

A segment of the Type Definition (DTD) file that is related to the morphological annotation is given in Figure 3.4.

3.3.2 Analytical annotation level

During the transformation of a sentence from the morphological to the analytical level, the corresponding linear sequence of words and punctuation marks is enriched with a *dependency structure* representing the given sentence. Each node of the structure is assigned with a so called *analytical function*. This structure is called an *analytic tree structure* (ATS).

Dependency structure

The basic principles of the dependency structure at the analytical level within the PDT can be formulated as follows ([Hajič–98]):

[1]Warning: distinguish between an SGML tag and a morphological tag. A morphological tag is an *element* in the terminology of the SGML, not the SGML tag!

```
<s id=cmpr9415:025-p19s2/bcc14zua.fs/#18>
<f cap>Ty<MMl>ty<MMt>PP2S1<MMt>PP2S5<MMl>ten<MMt>PDFP1<MMt> ...
<f>mají<MMl>mít<MMt>VPP3A<A>Pred<r>2<g>0
<f>pak<MMl>pak<MMt>DB<A>Adv<r>3<g>2
<f>někdy<MMl>někdy<MMt>DB<A>Adv<r>4<g>2
<f>takovou<MMl>takový<MMt>AFS41A<MMt>AFS71A<A>Atr<r>5<g>6
<f>publicitu<MMl>publicita<MMt>NFS4A<A>Obj<r>6<g>2
<D>
<d>,<MMl>,<MMt>ZIP<A>AuxX<r>7<g>8
<f>že<MMl>že<MMt>JS<A>AuxC<r>8<g>6
<f>to<MMl>ten<MMt>PDNS1<MMt>PDNS4<A>Sb<r>9<g>13
<f>dotyčnou<MMl>dotyčný<MMt>AFS41A<MMt>AFS71A<A>Atr<r>10<g>11
<f>kancelář<MMl>kancelář<MMt>NFS1A<MMt>NFS4A<A>Obj<r>11<g>13
<f>prakticky<MMl>prakticky<MMt>DG1A<A>Adv<r>12<g>13
<f>zlikviduje<MMl>zlikvidovat<MMt>VPS3A<A>Obj<r>13<g>8
<D>
<d>.<MMl>.<MMt>ZIP<A>AuxK<r>14<g>0
```

Figure 3.3: A segment of a SGML tagged sentence.: *"Ty mají pak někdy takovou publicitu, že to dotyčnou kancelář prakticky zlikviduje."* (The second line is shortened.)

- the analytical structure of the sentence is an oriented, acyclic graph with one entry node; the nodes of the tree are annotated by complex symbols (attribute-value pairs),
- the number of nodes of the graph is equal to the number of words in the sentence plus one for the extra root node.

In a dependency tree (see an example in Figure 3.5 (b)), the position of the word with respect to the vertical axis corresponds to the dependency relation among words in the sentence. For each edge, the upper word is *governing* and the lower one is *depending* (it completes, modifies, alters the upper word). The difference between analytic (surface) and tectogrammatical ("real") dependency structures will be discussed later in this chapter.

```
<!ELEMENT MMl - O (#PCDATA & R? & E? & e? & T* & MMt*)
                -- lemma (base form), description see the l tag;
                   machine assigned (by a morphological analysis program),
                   NOT disambiguated
                   -->
<!ELEMENT MDl - O (#PCDATA & R? & E? & e? & T* & MDt*)
                -- lemma (base form), description see the l tag;
                   machine assigned (by a tagger), disambiguated
                   if more than 1: n-best
                   -->
. . .
<!ELEMENT MMt - O (#PCDATA)
                -- morphological tag(s) as assigned by morphology,
                   NOT disambiguated
                   -->
<!ELEMENT MDt - O (#PCDATA)
                -- morphological tag(s) as assigned by machine, disambiguated,
                   possibly also with weight/prob; if more than 1: n-best
                   -->
```

Figure 3.4: A segment from the Document Type Definition File which corresponds to the morphological annotation.

For the sake of comparison, let us recall the other possibility of depicting the syntactic structure of a sentence. It is called a *derivation tree* (e.g., in [Melichar–97]) and it is related to a view of formal grammars going back at least to Chomsky's work in the 1950's. An example of a derivation tree is in the Figure 3.5 (a).

The key difference between dependency and derivation tree structures is that the former represent the product of the derivation, while the latter represent the derivation history. Dependency trees also directly reflect the head/dependent binary relations (head/dependent asymmetry) between lexical elements, which makes them closer to the semantic structure than the traditional derivation trees in which this asymmetry is not reflected.

Analytical function

An analytical function determines the relation between the dependent node and its governing node, or, in other words, the function of the dependent node with respect

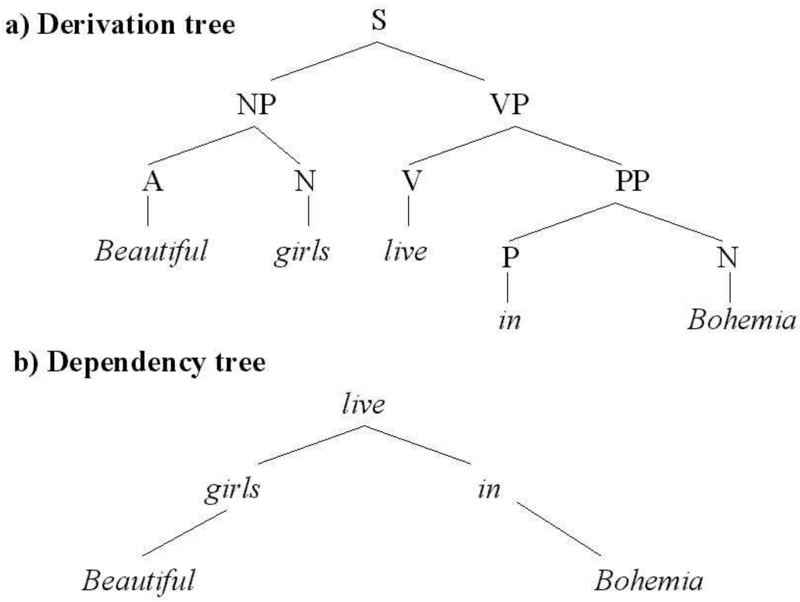


Figure 3.5: Derivation and dependency tree of the sentece *"Beautiful girls live in Bohemia"*.

to its governing node. The name of the node attribute bearing the analytical function is `afun`.

Let us mention several possible values of `afun`:

- `Pred`, predicate if it depends on the tree root
- `Sb`, Subject
- `Obj`, Object
- `Adv`, Adverbial
- `Atr`, Attribute
- `Pnom`, Nominal predicate's nominal part, depends on the copula "to be".

The representation of the sentence at the analytical level can be again stored in SGML format. In Figure 3.6 there is the corresponding segment of the DTD file.

Now, the reader can look at the segment of a SGML tagged sentence in Fig. 3.7 with deeper understanding.

An example of the analytical tree structure is depicted in Figure 3.8 (a).

```
<!ELEMENT A - O (#PCDATA)
                -- analytical (syntactic) function manually assigned
                   to the word form
                   in the f tag; for allowed list see annotator's guidelines
                   at fairway.ms.mf.cuni.cz / Projects / Treebank / guide
                   -->
<!ELEMENT MDA - O (#PCDATA)
                -- analytical (syntactic) function(s) as assigned by machine,
                   disambiguated, possibly with weight; if more than 1: n-best
                   -->
. . .
<!ELEMENT g - O (#PCDATA)
                -- governing node on the analytical level. For description
                   se also annotator's guidelines
                   at fairway.ms.mf.cuni.cz / Projects / Treebank / guide.
                   Text contents points to identical r elsewhere in the
                   same sentence. Pointer to node 0 is allowed - it is
                   the artificial root node added to each sentence
                   annotated on the analytical level.
                   This is the manually assigned gov. node.
```

Figure 3.6: A segment from the Document Type Definition File which corresponds to the morphological annotation.

3.3.3 Tectogrammatical annotation level

The annotation on the tectogrammatical[2] level results in so called *tectogrammatical tree structures* (TGTS). If an ATS reflects the surface syntactic structure, then a TGTS corresponds to the underlying sentence representation.

The transition from the ATSs to the TGTSs (described by Böhmová and Hajičová in [BH–99]) consists of two phases:

1. automatic pre-processing,
2. manual correction and the completion of the results of the first phase using user-friendly software.

During the transition from ATSs to TGTSs, the topology of the tree is slightly

[2] Both halves of the word "tectogrammatical" are of Greek origin: $\tau\epsilon\kappa\tau\omega\nu$ means builder, constructor, $\gamma\varrho\alpha\mu\mu\alpha$ means letter. The term "tectogrammatical representation" was introduced by H.B. Curry as the representation signifying how expressions represent process of construction.

```
<f>mají<MMl>mít<MMt>VPP3A<A>Pred<r>2<g>0
<f>pak<MMl>pak<MMt>DB<A>Adv<r>3<g>2
<f>někdy<MMl>někdy<MMt>DB<A>Adv<r>4<g>2
<f>takovou<MMl>takový<MMt>AFS41A<MMt>AFS71A<A>Atr<r>5<g>6
<f>publicitu<MMl>publicita<MMt>NFS4A<A>Obj<r>6<g>2
<D>
```

Figure 3.7: A segment of a SGML tagged sentence. The analytical function is bold-faced.

changed. For example, *synsemantic words* (functional words, nodes "without their own meaning"), e.g., prepositions, auxiliaries, subordinating conjunctions, as well as punctuation marks, are pruned, i.e., they do not have their own node in TGTS, but they are captured in the attributes of the remaining nodes representing the *autosemantic words*.

The transition from ATSs to TGTSs involves also the assignment of the *tectogramatical function*—so called *functor*, to every node in the tree. Functors are the tectogrammatical counterparts to the analytic functions.

There are approximately 60 functors divided into two subroups:[3]

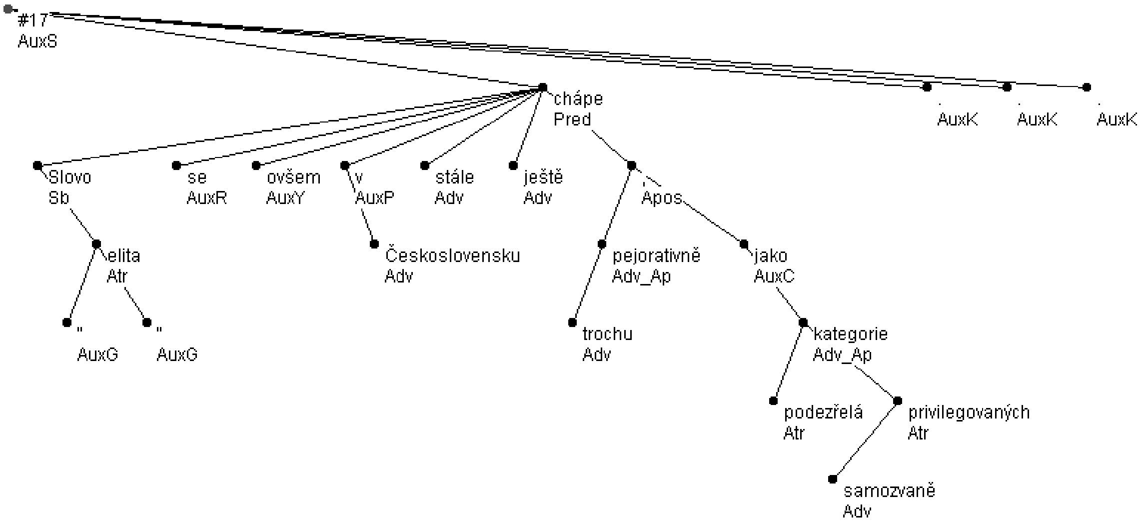
- *actants* : ACTor, PATient, ADDRessee, EFFect, ORIGin
- *free modifiers*: TWHEN (time-when), LOCaction, EXTent, BENeficiary, MEANS, ATTribute ...

Actants are the basic participants in the sentence, they are usually dependent on the verb node. The actants play a role of (often obligatory) "parameters" of the governing node. Among the nodes with a common governing node, there can be at most one actant of each type (e.g., there can be maximally one Actor in a sentence, though it can be expressed by coordination containing more words).

Free modifiers (circumstantials) describe modifications of the governing node. There can be more nodes with the same functor sharing the same governing node.

[3] Authentic examples of the usage of functors can be found in Appendix B.

(a) Analytical tree structure



(b) Tectogrammatical tree structure

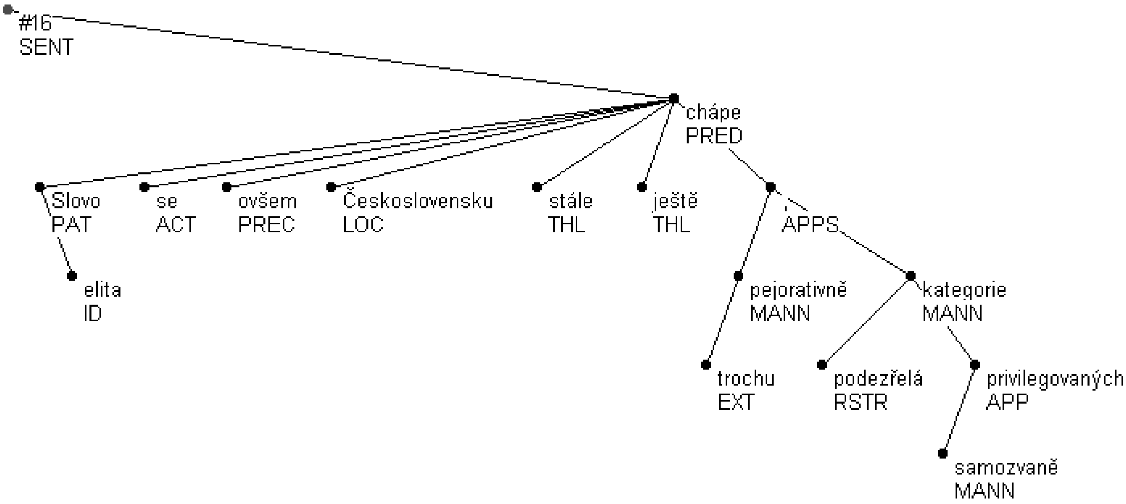


Figure 3.8: Analytical and tectogrammatical tree structures of the sentence "*Slovo "elita" se ovšem v Československu stále ještě chápe trochu pejorativně, jako podezřelá kategorie samozvaně privilegovaných...*" (*The word "elite", however, in Czechoslovakia still is understood a little pejoratively, as a suspicious category of self-appointed privileged people...*)

For instance, the sentence *"In Bulgaria we lived in tents"* can be analyzed as containing two spatial circumstantials LOC, both dependent on the node *"lived"*.

The reader can compare the analytical tree structure in Figure 3.8 (a) with the corresponding tectogrammatical tree structure in Figure 3.8 (b). More examples can be found in Appendix C.

Chapter 4

Automatic Functor Assignment: Problem Analysis and Data Preprocessing

Presently, the procedure of Böhmová *et al.* [BPS–99] solves automatically the topological conversion and the assignment of a few functors (e.g., ACT, PAR, PRED) during the transition from ATSs to TGTSs. However, most of the functors have to be assigned manually. The amount of labor involved in the manual annotation obviously slows down the growth of the PDT on the tectogrammatical level. Decreasing the amount of manual annotation has been the motivation for developing the more complex *automatic functor assignment* (AFA) system, the description of which forms the core of this diploma thesis.

4.1 Formulation of AFA problem

Supposing that the topological conversion of the ATS towards the TGTS has been done, the aim of the AFA is to attach a functor to every node of the tectogrammatical tree structure (or to as many as possible).

Since there is only a finite set of possible functors and all of them are known[1] in advance, we can formulate the problem of the AFA as *the classification of TGTS's nodes into 60 classes.*

In order to create a system which would be really helpful to human annotators, it has to fulfill several requirements:

- as many functors as possible should be assigned correctly,
- it must run in a reasonable CPU time, without any special hardware,

[1] The question of what is the ideal set of functors has probably not been completely closed yet, but no considerable changes occurred during the period of my work on this project.

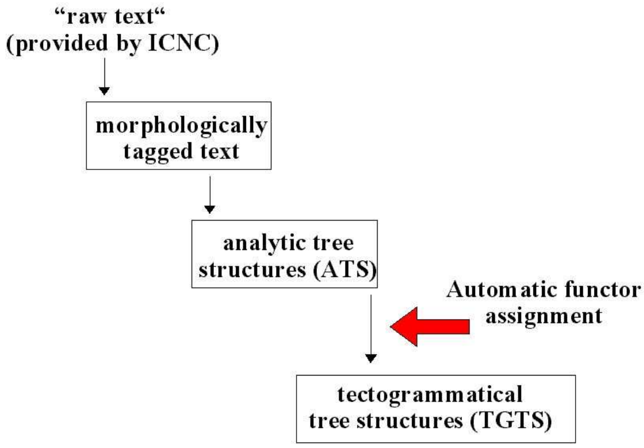


Figure 4.1: The position of the AFA system within the PDT project.

- it must be easy to apply, requiring no human interaction during the run time,
- it must make use of the background knowledge and all available data sources,
- it must be open, i.e., the amount of work for integrating new components or sources in the future should be minimized.

4.2 Initial situation

Let us briefly describe the situation in which the development of the AFA system started:

- *No general unambiguous rules* for functor assignment are available, human annotators use mostly only their language experience and intuition. We can never reach 100 % precision of the AFA system since even the *results of individual annotators sometimes differ.*[2]
- The annotators usually decide on the basis of *the whole sentence context*, and possibly even extra-sentential context. It was not measured how often it is

[2]This observation shows that the distinctions among the existing functor classes is probably not sufficiently sharp. When classifying the nodes into the functor classes, we should keep in mind Wittgenstein's aphorism: *"To remove vagueness is to outline the penumbra of a shadow. The line is there after we have drawn it, and not before."*

really unavoidable to consider the full context and how large this context must be. For the realization of the AFA system, it is practical to minimize the size of the context taken into account.

- Preliminary measurements revealed that *the distribution of functors is extremely non-uniform.* The 15 most frequent functors cover roughly 90 % of nodes (Figure 4.2). Conversely, there are hardly any examples for the rarest functors.

- It would be very time consuming to test the performance of the AFA system on randomly selected ATSs and find errors manually. Fortunately, we can use the ATSs for which manually created TGTSs are already available for initial tests, annotate them automatically and compare the results with the manually annotated TGTSs.

- The available TGTSs contain *imperfect data.* Some errors are inherited from ATSs, and functor assignments are in some cases *ambiguous* (nodes with more than one functor) or *incomplete* (some nodes have no functor yet). This again means that a 100% coverage cannot be obtained.

- The set of available TGTSs is too small. It cannot be viewed as a representative sample of the Czech language, many language phenomena do not occur in it at all.[3]

- There is no tag for idiomatic expressions in PDT yet, therefore they cannot be automaticaly extracted and analyzed now. For instance, the noun "*dobrota*" in "*sekat dobrotu*" (lit. "to make good", "to behave well") can not be viewed as a Patient, although it is a noun in accusative that is dependent on the verb in the active voice.

[3]But this is the deal of corpus linguists, they have to live with permanent doubts about the corpora. Let us cite the headlong attack of Noam Chomsky: *"Any natural corpus will be skewed. Some sentences won't occur because they are obvious, others because they are false, still others because they are impolite. The corpus, if natural, will be so wildly skewed that the description would be no more than a mere list."* ([AA-91]).

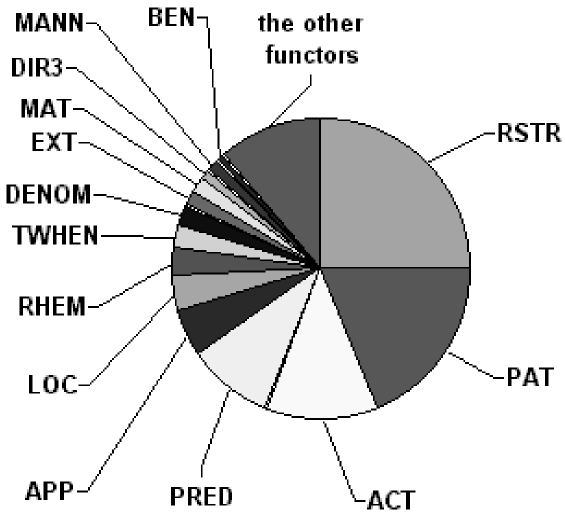


Figure 4.2: The distribution of functors is nonuniform.

4.3 Granularity

If a program is to decide what the correct functor of a node is, it must be provided with sufficient information for such a decision. Naturally, from the implementation point of view it is desirable to minimize the amount of the required information. The question of what such a minimal sufficient amount of information is, decomposes into two subquestions:

- what is the minimal neccessary size of the neigbourhood of the node in the tectogrammatical tree structure (the minimal tree context) which suffices for a unique determination of the functor, and
- what kind of information (which node attributes) contained in the minimal tree context has to be taken into account, and what can be neglected.

The first subquestion resembles the problem known from the area of parallel programming: how large "pieces" of the task can be solved separately; that is why I use the term *granularity* here as well.

I have already mentioned that the human annotators always analyze the entire sentence (and this is also the trivial upper bound of a context size, see Figure 4.3 (a)), without thinking of any subdivision into subtrees. But when trying

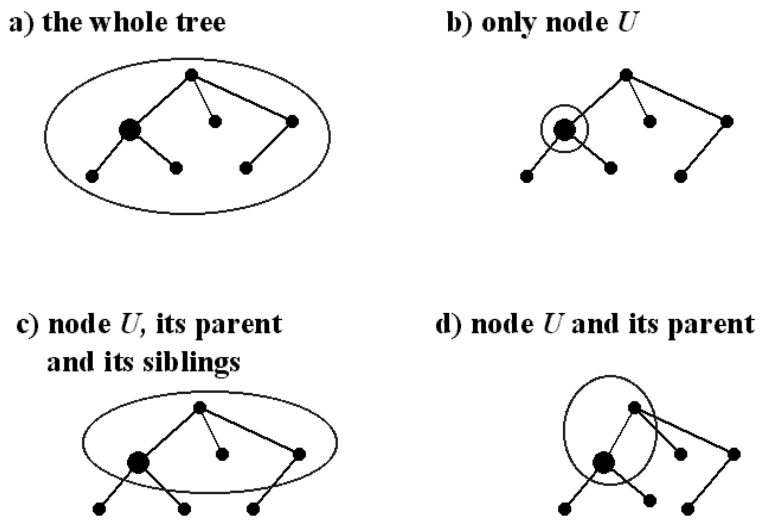


Figure 4.3: The minimal context of a node U.

to assign the functors automatically, we would have extreme difficulties with an implementation of such an approach, since:

- there is no justifiable natural limit for the size of a TGTS (measured by the number of nodes),
- there are nearly no limitations on the topology of the TGTSs.

The topology can be both very "flat" or very "deep". Many circumstancials might depend on one verb node, the sentence *"she surely goes with Peter to the cinema today at eight"* contains 5 offsprings of the node corresponding to *"goes"*. On the contrary, natural language can form pretty deep trees, for instance due to its recursive nature: *"the chair which was produced in a factory that employs 200 workers who are ..."*.

A further motivation for the effort to minimize the necessary context is based on the intuitive expectation that the mutually very distant nodes in the TGTS do not influence one another.

The trivial lower bound of the context size is only the node itself (Figure 4.3 (b)). This is obviously not enough. The node containing the expression *"po otci"*/ *"after father"* (let us recall that the preposition has been merged into the autosemantic node), can occur with at least three different functors, depending on the context:

1. *"Přišel po otci." ("He came after the father.")*, time circumstancial TWHEN,

2. *"Jmenuje se po otci." ("He is named as the father.")*, the functor NORM,

3. *"Zdědil majetek po otci." ("He inherited the property from the father.")*, the functor HER.

Another possible immediate context consists of the neigbouring nodes of a node to be assigned in the TGTS, i.e., its mother, its sibling(s), and its daughter(s), if any. Clearly, the governing node cannot be omitted when deciding about the correct functor as it was proved in the example above. On the other hand, it is very difficult to find an example where the knowledge of a child node is essential. Although it is evident that the depending nodes alter the meaning of the node itself, the change is mostly not so significant to make the value of the functor attribute crossing the border between functor classes. Therefore I will neglect the impact of the depending nodes.[4]

Since I do not suppose that the nodes with depth (the distance from the root) differing at least by two from the depth of a given node, are of any considerable influence of the functor, the last decision remains. Do the siblings of the given node (i.e., the nodes with the common governing node) bear any essential information (Figure 4.3 (c))? This is not the case, or at least not frequently (I did not find any example of such a situation).

The conclusion is that the only two inevitably remaining nodes are the node itself and its governing node (Figure 4.3 (d)). In other words, the attributes of the node itself and of its parent (mostly) provide a sufficient amount of information for the functor assignment of the former one.

[4]To be sincere, I have to admit I have later found several expressions, which I suspect of being counterexamples. For instance, the functor of the node *"v úseku"* can be either TWHEN in case of *"v úseku života"* or LOC in *"v úseku dálnice"*, thus being influenced by the depending node. In spite of the fact that such a situation is in the Manual for annotators ([Manual2]) solved using a special type of prepositions (e.g., *v průběhu čeho/during the process of something*) and therefore they do not have a node of their own in TGTS, one could probably find such a sentence where the knowledge of the dependent node would be important for functor assigning and it cannot be classified as a special preposition.

4.4 Feature selection and extraction

Now, let us return to the second subquestion from the beginning of the previous section: what kind of information from the minimal context (which attributes of the nodes in the context) has to be taken into account. Since during the analytical and morphological tagging, more than twenty attributes can be attached to every node, the selection of the most informative ones (*feature selection*) has to be done.

I selected the following ten attributes:

- for both the current node (the node to be assigned) and for its parent: *word form, lemma, full morphological tag, analytical function,*
- *the functor of the lower node,*
- *the preposition or subordinating conjunction binding the governing and the lower node.*

Three more attributes have been extracted from the morphological tags (*feature extraction*):

- part of speech of both nodes,
- case of the lower node.

The functor attribute has been selected just for the training and testing purposes of the AFA system. It cannot be used in the real-world application of the AFA system, since in such a case the functor is obviously unknown.

There are formal procedures how to select the most informative attributes (e.g., in [Kotek et al.–80]). I have selected the important attributes more or less on the basis of my intuitive judgements. Moreover, many attributes which did not get through the selection sieve were only of technical nature (identifiers, reserved attributes etc.).

Altogether, for each node of the TGTS—except for the auxiliary root—we have a vector of 13 symbolic (i.e., not numerical) attributes.

The task of the AFA can be now approximated as the classification of these vectors with twelve attributes.[5]

[5] The attribute containing the tectogrammatical function can be used just for the comparison of results, not as an input for the classification.

4.5 Data preprocessing

The sentences contained in the PDT are divided into files, each file has up to 50 sentences represented by trees. To assign functors to a tree structure means to assign a functor to each node in it (except to the root). For each node there is a corresponding vector of symbolic attributes. So the first preprocessing step is to transform each file of 50 trees into the sequence of vectors.

Besides the 13 attributes obtained by feature selection and extraction, two additional attributes are added to each vector: the name of the file where the tree is located and the ordinal number of the tree within the file. This is because once some phenomenon is observed in the preprocessed data, it is useful to know its location in the input data.

The output file is in plain text format, each line containing one vector with 15 attributes (separated by a tabulator) in this order:

1. The name of the original file.
2. The number of the sentence within the file.
3. The word form contained in the governing node.
4. The lemma of the governing word.
5. The full morphological tag of the governing word.
6. The part-of-speech of the governing node word, extracted from 5.
7. The analytical function of the governing node.
8. The word form of the node to be assigned.
9. The lemma of the node to be assigned.
10. The morphological tag of the node to be assigned.
11. The part-of-speech of the node to be assigned, extracted from 10.
12. The case of the node to be assigned, extracted from 10, or zero.
13. The preposition or subordinating conjunction binding the two nodes, or the empty string.

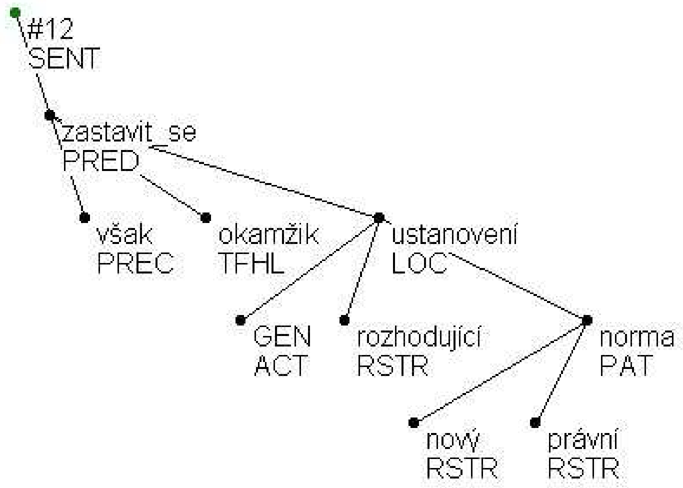


Figure 4.4: Example of the TGTS for the sentence *"Zastavme se však na okamžik u rozhodujícího ustanovení nové právní normy."*

14. The analytical function of the node to be assigned.

15. The functor of the node to be assigned.

The second preprocessing step is the elimination of those vectors where the functor is not specified or where it is specified ambiguously, for such data can be used neither for the training nor for the testing of the AFA system.

The last preprocessing step is the substitution of Czech accents by the corresponding letter without accent followed by underscore.[6]

A sample of the preprocessed data is shown in Figure 4.5, it corresponds to the TGTS in Figure 4.4 (the columns has been manually aligned for the sake of better readability):

4.6 Available material, training and testing set

When I started working on the AFA, 18 files with TGTSs were available. Since more new files become available relatively slowly (in each file, hundreds of functors

[6]This step produces ambiguity of `e_` and `u_`, but it has no (serious) impact on the quality of assigned functors. For instance, although the words *běžně* and *běžné* (commonly/common) are translated to `be_z_ne_`, they can be still distinguished using their morphological tags. Moreover, most of the proposed methods of the AFA do not use the lexical attributes of nodes.

```
bcb01trz 12 zastavme    zastavit1   vmp1a  v pred okamz_ik    okamz_ik    nis4a  n 4 na advt fhl
bcb01trz 12 zastavme    zastavit1   vmp1a  v pred ustanoveni_ ustanoveni_ nns2a  n 2 u  adv  loc
bcb01trz 12 normy       norma       nfs2a  n atr  nove_       novy_       afs21a a 0    atr  rstr
bcb01trz 12 normy       norma       nfs2a  n atr  pra_vni_    pra_vni_    afs21a a 0    atr  rstr
bcb01trz 12 ustanoveni_ ustanoveni_ nns2a  n adv  normy       norma       nfs2a  n 2    atr  pat
```

Figure 4.5: A sample of data (corresponding to the TGTS in Figure 4.4) after preprocessing.

have to be manually assigned), I did not wait for a larger data set.

I needed as much TGTSs as possible for data mining (for creating a list of adverbs etc.). On the other hand, it was necessary to leave some data untouched for the comparison purposes and for measuring the quality of the AFA system. Therefore I decided to (disjunctively) split the available files into a *training set* and a *testing set*.

The testing set consists of 3 randomly chosen files[7], consisting of 1089 testing vectors. The training set contains 15 files of TGTSs, after preprocessing containing altogether 6049 training vectors.

[7]Chosing several entire files was probably not the best decision, because the testing set can thus be biased: The sentences in one file are taken from the same original text, i.e., they have the same author and are related to the same topic, moreover they were annotated by the same annotator. It would have been better to select, say, 20 % from the whole set of edges, instead of selecting the whole files.

Chapter 5

Components of the AFA System

There is no simple and straighforward method to assign the functors at the tectogrammatical level automatically. Therefore, it was inevitable to look at the problem from different viewpoints and to combine a spectrum of various approaches. The "final" version of the AFA system consists of 12 different methods. The methods are specialized, each method assigns only a subset of the functors to be assigned.

The methods can be classified into four classes: rule-based methods, dictionary-based methods, nearest vector approach, and machine learning approach. They are described in the following four subsections, respectively.

Though this classification is helpful in explaining the functionality of the AFA system, the classification is not inherent to the problem itself. Rather than being predicted in advance, it arose during the AFA's evolution.

5.1 Rule-based methods

The rule-based methods (RBMs) consist of simple hand written decision trees. In the premises of the rules, lexical attributes (word forms and lemmas) in the attribute vectors are disregarded. E.g., there is no difference between the sentences *"Your brother went to the theatre"* and *"Your dog slept on the grass"* as far as the RBMs are concerned.

In order to simplify the references to the individual methods in the rest of this thesis, each method is assigned a short identifier typeset using nonproportional letters.

Currently I have 7 methods, each of which has reasonable precision (the abbreviation "$\rightarrow$ X" stands for "the node is assigned the functor X"):

1. `verbs_active`: **if** the governing node is a verb in the active voice **then**
   - **if** the analytical function (afun) is subject, **then** → ACT
   - **if** afun is object and the case is dative **then** → ADDR
   - **if** afun is object and the case is accusative **then** → PAT
2. `verbs_passive`: **if** the governing node is a verb in the passive voice **then**:
   - **if** afun is subject **then** → PAT
   - **if** afun is object and the case is dative **then** → ADDR
   - **if** afun is object and the case is instrumental **then** → ACT
3. `adjectives`: **if** the node corresponds to an adjective **then**
   - **if** it is a possessive adjective **then** → APP
   - **else** → RSTR
4. `pronounposs`: **if** the node is a possessive pronoun **then** → APP
5. `numerals`: **if** the node is a numeral **then** → RSTR
6. `pnom`: **if** afun is PNOM **then** → PAT
7. `pred`: **if** afun is PRED **then** → PRED

Unfortunately, I found only several feasible rules in the manual for annotators [Manual2]. They are utilized in `verbs_active` and `verbs_passive`.

The remaining five methods are based on an inspection of the training set. I simply searched for the correlations between the functors and the values of the analytical function or morphological categories and on the basis of this I formed hypotheses. I accepted only those hypotheses which were not in contradiction with common sense or with my language experience.[1] Therefore the resulting set of the 7 rule-based methods is more or less independent of the training set.

On the other hand, I am aware of the fact that the potential of the correlations between the functors and the non-lexical attributes is broader. Many rules

[1] For example, it is not surprising that the possessive adjective mostly represents appurtenance APP, though this was a quite new and useful fact for me.

probably remain hidden to my eye due to my limited linguistic knowledge, or because of the fact that some phenomena did not occur in the training set at all or only in a statistically insignificant amount (once, twice) that does not justify any generalization.

5.2 Dictionary-based methods

In contrast to the rule-based methods, sometimes the lexical value of a node is the only key to the functor, and everything else (e.g., part-of-speech of the governing node, etc.) can be neglected. I use the term *dictionary-based methods* (DBMs), since I collected *dictionaries* of adverbs, subordinating conjunctions, and prepositions for this purpose.

Some interesting side products emerged during the development of DBMs. For example, I extracted from the training data some adverbs and subordinating conjunctions which were previously not included in the Manual for annotators ([Manual2]). Now they can be used for further improvement of the manual.

Subordinating conjunctions

A dictionary of subordinating conjunctions (SCs) is used by the method `subconj`. It contains 38 couples {subordinating conjunction, functor}.

The dictionary was created in several steps:

1. 40 distinct couples were extracted from the training set,
2. 69 couples from the manual for annotators [Manual2] were added (the union contained 90 different couples),
3. 38 *unambiguous* subordinating conjunctios were selected.

An SC is called unambiguous if the nodes which are tied to its governor via this SC have always the same functor. E.g., *"když"* (when) is not unambiguous for it can appear with the functors COND or TWHEN. Table 5.1 shows a sample of the dictionary of unambiguous SCs.

The method `subconj` detects whether a node is tied to its governing node through an SC. In such a case, the SC is searched for in the dictionary. If it is found, the corresponding functor is assigned, otherwise the functor remains unassigned.

SC	functor
a proto	CSQ
ač	CNCS
ačkoli	CNCS
ačkoliv	CNCS
aniž	COMPL
byť	CNCS
dokud	THL

Table 5.1: A sample from the dictionary of subordinating conjunctions.

Adverbs

The dictionary of adverbs is created in the same way. 267 couples {adverb, functor} from the manual for annotators are merged with 236 couples found in the training set. Altogether, this dictionary contains 456 different couples. After elimination of ambiguous adverbs, the resulting dictionary contains 290 adverbs. Some of them are shown in Table 5.2.

It is worth noting that the ambiguous adverbs were most frequently accompanied with the functor ATT (attitude) and with the functor MANN (manner), e.g., *"krásně"*,*"moudře"*. The co-occurence of these two functors in the extracted dictionary is so frequent that it opens the question whether the boundary between them is always sharp enough and whether it would not be better to establish one common functor instead of distinguishing them.

A sample from the dictionary of adverbs is in Table 5.2.

Prepositions and nouns

The method `prepnoun` is based on the fact that some nouns preceded by a given preposition are always accompanied by the same functor. The dictionary of this method consists of triples {preposition, noun, functor}. The dictionary was created in three steps:

1. all such triples were isolated from the training set (659 different triples),

2. the triples containing ambiguous preposition-noun couples were eliminated,

adverb	functor
nikdy	TWHEN
nikoliv	RHEM
nově	MANN
nutně	MOD
nyní	TWHEN
obchodně	MANN
obecně	EXT

Table 5.2: A sample from the dictionary of adverbs.

3. those triples which occur at least twice in the training set become included in the dictionary.

A sample from the dictionary of these triples is in Table 5.2.

preposition	noun	functor
v	roce	TWHEN
v	Praze	LOC
v	době	TWHEN
pro	podnikatele	BEN
od	doby	TSIN
do	vlastnictví	DIR3
ze	zisku	DIR1

Table 5.3: A sample from the dictionary of for the method `prepnoun`.

5.3 Nearest vector approach

The third approach used in the AFA system does not require any rules or dictionaries. It uses the training data directly as a source of information. When assigning a functor to a symbolic vector, we simply find the *nearest*, i.e., most similar, or closest, vector in the training set. Then we just take the functor of this most similar vector as the result. If we define a metric on the feature space, we can simply find the nearest vector with respect to this metric. Instead of a binary function representing the metric, I define a binary function representing the similarity of vectors[2], let us call it *similarity function* $s_{\vec{w}}(\vec{v}, \vec{t})$. Similarity and distance measures are two sides of the same coin, so, the most similar vector and the least distant one are the same, with respect to a given vector.

Let $\vec{v}$ and $\vec{t}$ be vectors from the symbolic feature space, $\vec{w}$ is the weight vector of non-negative real numbers representing the importance of individual attributes of the vector (the higher value, the more important), $e(a, b)$ is the equality function (if both arguments are equal then returns 1, otherwise 0), then the similarity function can be defined as follows:[3]

$$s_{\vec{w}}(\vec{v}, \vec{t}) = \sum_{i=1}^{12} w_i \cdot e(\vec{v}, \vec{t})$$

The function $f(\vec{t})$ which returns the functor corresponding to the vector is defined on the domain of the training set T. The functor assignment can be approximated as searching for the value of f outside T. If $\vec{v}$ is an unassigned vector, then its functor can be estimated as:

$$f(\vec{v}) = f(arg \max_{\vec{t} \in T} s(\vec{t}, \vec{v}))$$

Obviously, the vector $\vec{w}$ plays a crucial role for correct functor assignment. The weights have been approximated intuitively, taking into account, for example, the following facts:

- the weight of the preposition is higher than the weight of the word form of the governing node,
- the sum of weights of the governing node's lemma, preposition and case of

[2]The reason for talking about similarity rather than distance is only psychological: if two symbolic vectors have nothing in common, I prefer to say that their similarity equals zero instead of their distance equals infinity.

[3]I am aware of the fact that this concept of the similarity function is probably too simplistic with respect to the complexity of the problem; no combination of a few weight coefficients can reflect all the language phenomena which are important for the AFA system. That is why is I did not accented this method too much, though one could play with tuning the weight vector and try to astonish the audience using soft computing methods for its optimization, especially genetic algorithms or neural networks.

the dependent node is higher than the sum of the weights of the remaining attributes,

- the analytical function of the node to be assigned has higher weight than the weight of the part-of-speech, etc.

The functor assignment then looks for example as follows. There is a sentence in the testing set which contains the expression *zálohy na daně*. A functor is to be assigned to the dependent node *daně*. In the training set, the most similar record is found (*návrh na stanovení*) and the functor PAT of its lower node (*stanovení*) is used, which is correct.

The disadvantage of the nearest vector method is its black box behaviour. Besides tuning weights, there is no other way to incorporate some other background knowledge, and it is difficult to decide which language phenomena are rendered via weights.

By the way, the nearest vector method can be also viewed as a special case of machine learning —so called *case-based learning*—since the program takes advantage of the experience given as a set of instances solved in the past. It is incremental learning because new examples can be easily inserted into the training set.

5.4 Machine learning approach

In order to exploit the information in the training set as much as possible and to find some more rules for functor assignment, I decided to apply a ML approach. I have to emphasize that this would not have been possible without the help of Sašo Džeroski from the Jožef Stefan Institute[4] in Ljubljana.

We applied Quinlan's ML system C4.5. Speaking in terms defined in Section 2.3, C4.5 can be described as inductive, symbolic, supervised, multiple concept, non-incremental, TDIDT (Top Down Induction of Decision Trees) ML system. In other words, it takes a training set with known classification (i.e., with known functors) as an input and yields a decision tree as an output. C4.5 can also prune the tree in order to obtain simpler and more general rules; it also evaluates the quality of such a tree on a testing set.

[4] `http://www.ijs.si/ijs`

Having obtained the decision tree, I pruned it once more by hand in order to eliminate the leaves of the tree for which the expected precision is lower than 80 %. This is the reason for the identifier of this method being `ml80`.

5.5 Alternative and complementary approaches

In this section, several additional approaches to the AFA task will be outlined. For various reasons that will be given below, these other approaches have not been implemented in the present AFA system. Therefore, they will not be mentioned in the following two chapters any more. However, some of them could contribute to the quality of a future AFA system, either as an alternative stand-alone system or as an extension of the presented one. This is the reason why I discuss them.

5.5.1 Neural network

My first proposal (from September 1999) for solving the problem of the AFA was based on (Artificial) Neural Networks (NN, [MR–91]). A rough schema of such a system is depicted in Figure 5.1.

After feature selection from a TGTS, the selected information is encoded into a numerical vector, which is an input of a backpropagation NN with one inner layer. In the last layer, each neuron is related to one functor. The resulting functor corresponds to the neuron in the output layer with the highest output value.

The weights $w_{1,i}$ and $w_{2,j}$ can be estimated from the training data (supervised learning) by a method of backpropagation learning.

This approach was not implemented because of the following problems the implementation would have involved. Firstly, NNs can perform well especially in applications where the topology of the input data space is clear and where the notion of distance makes sense. Unfortunately this is not the case for the TGTSs; all the input data for the AFA system are symbolic (non-numerical). For example, it would be difficult to define distance within a set of lexical entries (within the set of adverbs, etc.). Therefore the translation of the symbolic features into a numerical form is not trivial. Secondly, the well-known black-box behaviour of NNs could cause difficulties when we would try to make use of any background knowledge (rules, etc.).

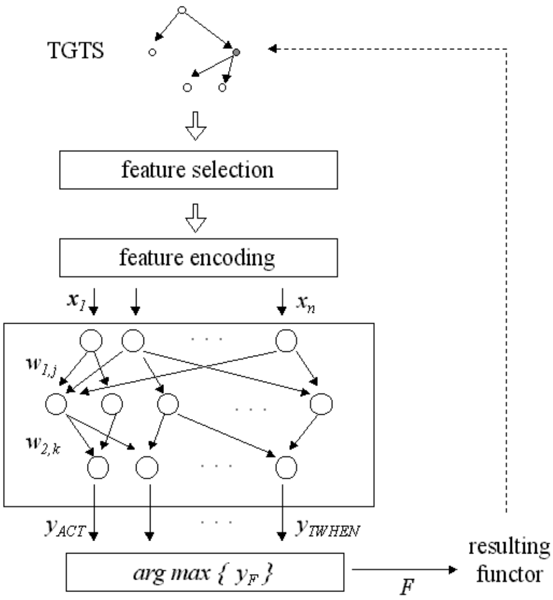


Figure 5.1: Sketch of a AFA system based on the backpropagation neural network.

5.5.2 EuroWordNet

The performance of the dictionary-based methods is naturally limited by the amount of lexical entries in the dictionaries. One of the linguistic resources, which one could use to improve the coverage of the dictionaries, is EuroWordNet[5].

EuroWordNet is a multilingual database with wordnets for several European languages (Dutch, Italian, Spanish, German, French, Czech and Estonian). The wordnets are structured in the same way as the American wordnet for English (Princeton WordNet) in terms of synsets (sets of synonymous words) with basic semantic relations between them. Among other relations, synonymy, hypernymy (relation to a more general word) and hyponymy (relation to a more specific word) are captured.

A basic idea of the data contained in a wordnet can be obtained from Figure 5.2, where the response of the Princeton WordNet[6] to the query about hypernyms of the word forest is depicted.

The basic version of the Czech WordNet can be bought from ELRA/ELDA[7].

---

[5] `http://www.hum.uva.nl/~ewn`

[6] `http://www.cogsci.princeton.edu/cgi-bin/webwn`

[7] European Language Resources Association (ELRA), European Language resources Distribu-

```
2 senses of forest

Sense 1
forest, wood, woods -- (the trees and other plants in a large densely wooded area)
 => vegetation, flora -- (all the plant life in a particular region)
   => collection, aggregation, accumulation, assemblage -- (several things grouped together)
     => group, grouping -- (any number of entities (members) considered as a unit)

Sense 2
forest, woodland, timberland, timber -- (land that is covered with trees and shrubs)
 => land, dry land, earth, ground, solid ground, terra firma -- (the solid part of the
    earth's surface; "the plane turned away from the sea and moved back over land";
    "the earth shook for several minutes"; "he dropped the logs on the ground")
   => object, physical object -- (a physical (tangible and visible) entity; "it was
      full of rackets, balls and other objects")
     => entity, something -- (anything having existence (living or nonliving))
```

Figure 5.2: WordNet 1.6 results for "Hypernyms (this is a kind of...)" search of noun "forest".

The number of lexical units in it is naturally much smaller in comparison with its older and bigger English cousin. The development continues further at the Department of Information Technologies, Faculty of Informatics, Masaryk University, Brno.

EuroWordNet could be used in the AFA system for example as follows. The dictionary of the method `prepnoun` (triples preposition-noun-functor) would be manually enriched with more general entries like {v (case=locative), fyzický objekt, LOC} ({in, physical object, LOC}). When assigning functors, all the hyponyms of the term "physical object" which are preceded by the preposition "v" could be assigned the functor LOC (spatial circumstancial). Thus also some previously unseen words could be assigned, e.g., "v lese" (in the forest).

This approach has not been implemented yet due to technical difficulties. The EuroWordnet database is distributed with a browser of the database, but not with a suitable interface for other programs that would enable automatic access to the the data.

tion Agency (ELDA), `http://www.icp.grenet.ft/ELRA/home.html` .

5.5.3 Matching Algorithm

Before the conception of the AFA system could get sharper contours, it was necessary to get a feeling for the real content of the data at the tectogrammatical level. This is why I took a sample of 20 tectogrammatical trees and I studied it carefully. I performed manual measurements of selected phenomenona, e.g., the distribution of nodes with respect to part-of-speech, the amount of nodes directly dependent on a verb node, the relative frequency of individual functors etc.

Having observed these characteristics of the data, I was able to estimate the trivial lower bound of the precision to be at least 40 % for the case when only very simple methods would be used. This was rather optimistic and encouraging news.

However, the aim of my thesis project was to reach at least the level of 70 % precision. For these purposes, I designed the following three-phase *matching algorithm* (Figure 5.3):

1. *Expected Roles*: in the first phase each non-root node generates (using all available information about itself) a set of possible functors—i.e., it suggests the possible tectogrammatical roles for itself. Each generated functor should be enriched by a weight which enables an ordering of these functors with respect to their frequency of occurence. For example, a node with the adverb "naopak" (on the contrary) generates only one functor (PREC) with the maximum weight, since it can play no other role.

2. *Expected Offsprings*: in the second phase each non-leaf node generates a set of functors which can possibly depend on this node, again accompanied by weights. Moreover, some requirements about the dependent node can be added. For instance, the verb "zamilovat se" (to fall in love) requires the functor PAT to be tied with the preposition "do" (to fall in love *with* somebody).

   It is important to note that more than one set of functors can be generated. This is the case of verbs which have more than one valency frame.

3. *Matching*: in the third phase, each non-leaf node matches its expectations against the possible roles of its offsprings. The aim is to *saturate* as many expected roles as possible and to fulfill all the requirements. When there are more possibilities of coupling, we prefer the one with the highest weight.

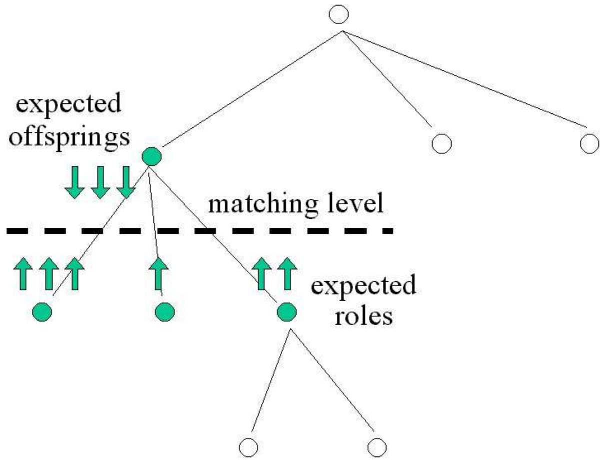


Figure 5.3: Matching algorithm.

For nouns, adjectives, pronouns, numerals, adverbs and verbs, I considered what their behaviour would be in each phase of the algorithm. I paid special attention to verbs, trying to profit from the concept of valency frames as it is formulated in FGD [Panev–80], or in a different form as L-valency frames in [Pala–99].

Though this approach seemed promising and it would have enabled a unifying solution of the problem of automatic functor assignment, I never implemented it. With respect to the uncertainity of the result, I found the amount of necessary work inadequate. I expected difficulties especially with the initial tuning of weights, the definition and implementation of the semantic distance function, incorporating the lexicon of valency frames, etc. Later, I decided to concentrate rather on implementing and testing a number of small specialized modules. This proved to be a more efficient way to do the job.

5.5.4 Valency frames of verbs

The term valency is in this context related to the ability of a word (especially a verb, but also a noun or an adjective) to "bind" other words.

One way of formulating, what a verb valency frame is, is as follows. The valency frame of a verb contains the arguments (obligatory or optional) it combines with, actants and/or free modifiers. For instance, the valency frame of the verb *otevřít* (*to open*) contains an Actant and a Patient. Every verb has at least one valency frame, though it can be empty (*pršet/to rain*).

For the purposes of the AFA, Karel Pala provided me with a valency dictionary of about 4400 of the most frequently occurring Czech verbs. In Figure 5.4 a sample from the dictionary is shown. Note that nominative arguments are not contained in this dictionary.

```
zadívat se do někoho(2)
zadívat se do něčeho
zadívat se na někoho(4)
zadívat se na něco
dohlédnout na někoho(4)
dohlédnout na něco
dohlédnout něčeho
zlevnit něco
zlevnit se v něčem
evokovat něco
konzumovat něco
nastřelit někoho(4)
nastřelit někoho(4) něčím
nastřelit něco
nastřelit něco něčím
```

Figure 5.4: A sample from the dictionary of verb valency frames.

My hypothesis was as follows. If the verbs could be automatically classified into less than 100 classes with respect to their valency frames, then it could be possible to manually complete these classes with functors and thus all these verbs would have their valency frames equipped with functors (the total number of all frames in this dictionary is about 28000, therefore it was not realistic to manually supply all the arguments in all the frames with their functors).

First, I preprocessed the dictionary. For each verb, I merged all its valency frames from the original list into a single frame. For example, there are four frames for the verb *přepadnout* in the original list: přepadnout někoho(4), přepadnout někoho(4) v něčem, přepadnout do něčeho, přepadnout přes něco. The resulting union of the frames is *přepadnout někoho(4) v něčem do něčeho přes něco*.[8].

[8] I am aware of the fact that after this step the alternative (mutually exluding) arguments of a verb may appear in one frame. However, I did not find any other automatic way to considerably reduce the number of frames per verb.

Next, the differences between animate and inanimate arguments were disregarded. For instance, both *někomu* and *něčemu* (animate and inanimate in the dative case) were rewritten to `#3`. Prepositional cases were substituted by the respective preposition followed by an underscore and the case expressed as a number, e.g., *v něčem* was rewritten as `v_6`. Only the 15 most frequent prepositional and direct cases were processed (`#4, #7, v_6, #3, na_4, do_2, na_6, z_2, k_3, s_7, po_6, #2, za_4, u_2, od_2`), all the remaining were ignored. A sample from the preprocessed dictionary is shown in Figure 5.5.

koordinovat	#4 s_7
kopat	#4 #7 do_2 za_4
kopnout	#4 #7 do_2
kopírovat	#4 od_2 z_2
korespondovat	#3 o_6 s_7
korespondovat si	s_7
korigovat	#4 #7 v_6
korunovat	#4 #7 na_4 za_4
koukat	#3 na_4 po_6 z_2
koukat se	do_2 na_4 po_6 z_2
kouknout	do_2 na_4 po_6 z_2
koupat	#4 v_6

Figure 5.5: A sample from the preprocessed verb valency dictionary.

Having this material in hand, I tested two methods of classification: binary classification tree and equivalence classes, which I discuss below.

Binary classification tree (top-down clustering). The set of all verbs was recursively divided into two parts according to occurence/non-occurrence of a selected prepositional or direct case. The case was chosen such that the difference in the sizes of the two created sets was minimal. Thus the resulting tree was as balanced as possible with respect to the weights of leaves, the weights being expressed as the number of verbs in the corresponding class. The lower bound for the size of a class was 20.

The binary classification tree was automatically induced from the dictionary described above; for this, I wrote a Perl program. All the verbs were classified into 76 classes. A fragment from the classification tree is depicted in Figure 5.6.

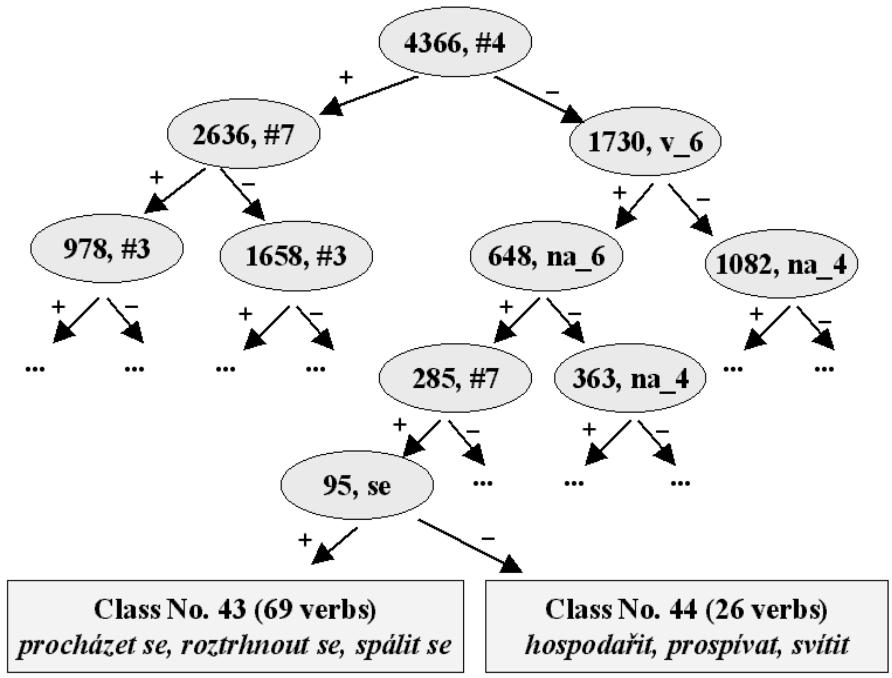


Figure 5.6: Binary classification tree of verbs with respect to their valency frames. In each non-leaf node, the number of verbs in the subtree and selected prepositional or direct case are inscribed. Left subtree contains always verbs that have this case in their valency frames, right subtree contains the rest. The leaf nodes represent the resulting classes.

Equivalence classes. An equivalence relation (symmetric, reflexive, transitive) on the set of verbs can be defined via the preprocessed valency dictionary: two verbs are "equivalent" if they have the same (preprocessed) valency frame. This equivalence relation entails a partitioning of the set verbs into the equivalence classes, i.e., all the verbs in one class have the same (preprocessed) valency frame. For example, one of the equivalence classes contains the verbs with valency frames containing only an object in the accousative case (*okupovat #4/occupy, pochopit #4/understand, prozkoumat #4/explore...*). The ten largest equivalence classes are in Appendix D.

The latter classification method seemed more promising, that is why I consulted it with Jarmila Panevová. According to her opinion, out of the 50 largest equivalence classes, only 4 can be uniquely assigned with functors. The conclusion from this experiment is that none of these classifications helps to assign functors to the verb valency frames. It is very likely that there is no other than manual (or semiautomatic) way to do it.

In the second experiment I concentrated only on the valencies typically realized by nouns in the genitive case tied with prepositions *z* (from) and *do* (into, till). In the valency dictionary, 1428 verbs with at least one of these valencies were found. The hypothesis is that they should mostly represent functors DIR1, resp. DIR3. From this set, I manually removed 114 verbs for which the hypothesis does not hold. (e.g., *"zamilovat se do někoho"/"to fall in love with somebody"* does not engage a directional modifier). I collected the remaining verbs into a dictionary of "directional-verbs". This dictionary was used in a new AFA module. This module assigned all nouns in the genitive case, which were dependent on a verb from the directional-verbs dictionary and which were accompanied by the preposition *z* (resp. *do*), with the functor DIR1 (resp. DIR3). I tested it on the union of the training and testing sets. After automatically removing a few nouns of clearly "non-directional" meaning (e.g., *"rok"/"year"*), 71 *z/do* remained genitives to be assigned. Using the directional-verbs dictionary, 49 functors were assigned, 43 of them correctly (precision 88 %, recall 0.6 %).

This approach was not incorporated into the presented version of the AFA system since the recall is too low. However, the potential of valency-based methods goes far beyond the processing of these two prepositions. Unfortunately, for the further development of the valency-based methods, adding functors into the verb valency dictionary manually seems to be inevitable.

5.5.5 Categorial grammar

The term Categorial Grammar (CG, cf. [Steedman-98] for a brief overview) names a group of theories of natural language syntax and semantics in which the main responsibility is borne by the lexicon. This is an alternative approach to Chomsky's Context-free Grammar. The lexicon associates a functional type or category with all grammatical entities. The category associated with a word captures (among others) two things: what are the categories of the words that are expected on the left- and right-hand side of the respective word and what is the resulting type of the whole after the saturation of these expectations. For example, the category of the word "*likes*" is (S\NP)/NP, since one noun phrase is required on the left-hand side (subject) and another on the right-hand side (object).

Very recently I have realized that the idea of the Matching Algorithm bears a slight resemblance with the main principle of CG: instead of cutting the sentence into phrases, a lexical element "generates" its expectations about its neighbourhood within the sentence and the expectations of the neighbouring elements have to meet each other.

The symbiosis of categories and a dependency approach based on the Prague School of Linguistics has been elaborated in the framework of Dependency Grammar Logic by Geert-Jan Kruijff (a introduction to DGL can be found in [Kruijff–99] or in [Kruijff–01]). This formalism is "tailored" for FGD and therefore is ready to be tested on the PDT data. The task of the AFA could be one of the possible applications of DGL.

Chapter 6

Implementation Details

6.1 Interface to the fs format

The files containing the tectogrammatical tree structures are saved in the so called fs format. This format was designed together with a general graph editor by Michal Křen ([Křen-96]). This editor provides a graphical user interface for comfortable work with graph structures (under MS Windows). In the PDT project, it is used for manual modifications of the trees (including, e.g., functor annotation) both on the analytical and the tectogrammatical level.

When trying to automatically assign functors, I need access to the contents of the fs files. For this purpose, I use an interface written by Petr Pajas which is composed of two parts:

- `foreach.pl` is a Perl script that reads another Perl script from standard input and executes it for every node of every tree in the file; it enables to read/write values of all attributes attached to a node,

- `hrany2.fsp` extracts for each node 15 attributes (as described in Section 4.5) and writes them to the standard output.

The training set (and similarly the testing set as well), i.e., the 6049 training vectors, can be saved into the file `train.txt` by executing the following pipeline:[1]

```
> foreach.pl ~/FUNKTORY/VstupniData/Train/*.fs <hrany2.fsp >train.txt
```

A piece of the output file is in Fig. 4.5 on page 34.

[1] The components of the AFA system were developed under the Linux environment.

6.2 Perl assigners

Following the common design practice of modularity, I prefered to compose the AFA system of a number of small Perl programs ("assigners"). Each assigns only some functors, and above all, each can be developed and tested independently on the remaining ones. Each method described in Chapter 5 has its own assigner.

In order to be able to glue all the modules together later, I formulated the following mandatory rules for the assigners:

1. The input data are to be read from the standard input, each line corresponding to one vector, the 15 columns are separated by tabulator. The columns are ordered in the same way as in the list on page 4.5.

2. If an assigner can "guess" what the correct functor is, then it attaches the functor into the 16th column and the assigner's "signature" starting with the "&" into the 17th column; the line with these 17 columns is written to the standard output.

3. If an input line is already assigned, then the line is copied to the standard output without any change, i.e., once assigned functor is never overwritten.

4. If an input line is not assigned yet and the assigner cannot assign the functor, then the line is copied without any change.

This stategy brings several advantages. First, by reordering the assigners (simply by changing their order in the pipeline, without changing any single line of Perl code) in such a way that the assigners with a higher precision are applied first, the overall precision is improved. Second, it is easy to add a new assigner, or to remove an assigner, e.g., one with low precision. Third, we can monitor not only the performance of the whole system, but also the performance of individual assigners separately.

All the assigners have one common template, only the subscribing string and the decision part (lines, where the functor is computed) are varying. The following assigner corresponds to the rule-based method `verbs_active`:

```
#!/usr/bin/perl

$subscribe="verbs_active";
```

```
while (<>)
{
    if (m/&/)
    { print }
    else    {
    $functor="";
        chop;
    @_=split("\t");

#----------------- DECISION PART  - BEGINNING
        if ((@_[4]=~m/^v[^s]/) && # is the governing node a verb in active form and
        (@_[12] eq "")) # is the given node tied without preposition nor conjunction?
    {
        if (@_[13] eq "sb") {$functor="act"}
        elsif (@_[13] eq "obj")
        {
            if (@_[11] eq 3 ) {$functor="addr"}
            elsif (@_[11] eq 4) {$functor="pat"}
        }
    }
#------------------ DECISION PART - END

    if ($functor eq "") {print "$_\n" }
    else {print "$_\t$functor\t\&$subscribe\n"}
    }
}
```

In the case of a dictionary-based method, the dictionary has to be loaded into an associative array first. In the decision part, the lexical value is searched in the associative array:

```
#----------------- DECISION PART  - BEGINNING
    if (@_[10] eq "d")  # is it an adverb?
    {
       $functor=$adverbs{@_[8]}
    }
#------------------ DECISION PART - END
```

In the nearest vector assigner (signature `similarity`), the whole training set is loaded into the array called `pole`. The similarity function is implemented in the decision part and the nearest vector is found:

```
#----------------- DECISION PART  - BEGINNING
$max=0;
    for ($i=0;$i<=$count;$i++)
    {
        $weight=0;
        @tr=split(/:/,$pole[$i]);

            if (@_[8] eq @tr[8]) {$weight+=15; };   #lower lemma
            if (@_[9] eq @tr[9]) {$weight+=18};    #lower tag
            if (@_[11] eq @tr[11]) {$weight+=60};   #lower case
            if (@_[10] eq @tr[10]) {$weight+=49};   #lower PoS
            if (@_[13] eq @tr[13]) {$weight+=50};    #lower afun
            if (@_[12] eq @tr[12]) {$weight+=60};    #preposition or conjunction
            if (@_[6] eq @tr[6]) {$weight+=30};   #upper afun
            if (@_[3] eq @tr[3]) {$weight+=10};   #upper lemma
            if (@_[4] eq @tr[4]) {$weight+=12};   #upper tag
            if (@_[5] eq @tr[5]) {$weight+=30};   #upper PoS

        if ($weight>$max) {$max=$weight;$functor=@tr[14]}
    }
#------------------ DECISION PART - END
```

6.3 Machine learning

The assigner based on machine learning was created in 5 steps:

1. Different feature selection and extraction; I restricted the set of attributes which are in the input vectors for C4.5; I omitted attributes containing word forms and lemmas of the governing and the dependent nodes (and, of course, also the name of the source file and the ordinal number of the sentence); instead of taking the whole morphological tags (as described on page 15), only their prefixes were extracted.

2. Preparation of input files for the C4.5; a file containing the list of possible values of all attributes, and files with training and testing set were trans-

formed into a format required by the C4.5; this is a sample from the training file:

```
n, n, adv, a, a, 0, null, atr, rstr.
vs, v, obj, n, n, 2, do, adv, dir3.
vp, v, pred, vs, v, 0, z_e, obj, pat.
znum, z, sb, dg, d, 0, null, auxz, ext.
n, n, atr, znum, z, 0, null, sb, rstr.
```

3. The C4.5 was applied on the prepared data.
4. The leaves with lower than 80 % expected precision were pruned.
5. The resulting decision tree was semi-automatically translated into Perl code.

A sample from the file with the learned decision tree in text representation is depicted in Figure 6.1.

```
dep_afun = sb:
|   gov_pos = a: rstr (1.0/0.8)
|   gov_pos = j: pat (1.0/0.8)
|   gov_pos = n: rstr (21.0/8.0)
|   gov_pos = null: act (1.0/0.8)
|   gov_pos = z: act (19.0/5.9)
|   gov_pos = v:
|   |   gov_morph = vp: act (463.0/25.9)
|   |   gov_morph = vr: act (133.0/12.9)
|   |   gov_morph = vs: pat (28.0/8.2)    *
|   |   gov_morph = vf:
|   |   |   dep_case = 0: pat (2.0/1.0)
|   |   |   dep_case = 1: act (6.0/3.3)
|   |   |   dep_case = 4: pat (1.0/0.8)
```

Figure 6.1: A sample from the file with the text representation of the learned decision tree.

It is interesting to observe that the machine learning approach learns also some rules which are part of the manual for annotators [Manual2]. For instance, the line in Figure 6.1 that is marked with an asterisk corresponds to the following rule from the manual: if a subject is dependent on a verb in the passive voice, then its functor is PAT. This observation proves that the C4.5 can really uncover some simple rules that are valid in the training set. Besides those specified in the manual, it also learns "new" regularities.

This is a fraction of the semiautomatically created Perl code of the assigner `ml80` which corresponds to a part of sample in Figure 6.1.

```
if ($dep_afun eq "sb") {
     if ($gov_pos eq "v") {
         if ($gov_morph eq "vp") {$functor="act"};
         if ($gov_morph eq "vr") {$functor="act"};
     }};
```

If the analytical function of a node is Subject and its governing node is a verb in the active voice, then this code assigns the functor ACT to the dependent node.

6.4 Auxiliary tools

It proved to be very useful to have a few tools for exploring the automatically assigned data (i.e., the stream of rows with 17 columns). I mention only a few of them:

- `correct.pl` and `incorrect.pl`, Perl filters extract either the correctly or incorretly assigned lines,
- `assigned.sh` and `unassigned.sh`, shell filters extract either the lines where the functor has been automatically assigned, or where it was not,
- `stat.pl` performs a statistic evaluation of the qualitative characteristics of the performed functor assignment.

The tools can be further combined with shell commands, e.g., if I want to know what are the most frequent misclassifications, I send the assigned data into the following pipeline:

```
incorrect.pl | cut -f15,16 | sort | uniq -c | sort -nr | head
```

and thus obtaine for example the following missclassifications and the number of their occurences:

```
16    pat   act
14    app   pat
11    ev    twhen
10    pat   app
7     id    rstr
```

This is useful for determining where the AFA system makes the most errors, and thus where further improvements are needed.

6.5 SQL queries

Sometimes I employed the Simple Query Language (SQL), especially when it was necessary to interconnect more files. For example, I had a file with a single list containing only unambiguous adverbs and a file with two columns: adverbs (both ambiguous and unambiguous) and functors. The task was to find a correct functor for each unambiguous adverb, i.e., to construct the dictionary of adverbs as defined in Section 5.2. For this, I transformed the files into the tables `AllAdverbs` and UnambigAdverbs and executed the following query:

```
SELECT All_Adverbs.Word, All_Adverbs.Funktor
FROM Unambig_Adverbs
INNER JOIN All_Adverbs ON Unambig_Adverbs.Word = All_Adverbs.Word;
```

6.6 Gluing the components together

There are two possibilities where to store the assigned functors:

1. into the text file as the 16th column (as it was described in the Section 6.2); this is used only for development and testing purposes,
2. directly into the original file in fs format; this is used for the automatic pre-annotation of the files for annotators.

In the former case, all the data goes through a long pipeline, e.g., through the following pipeline:

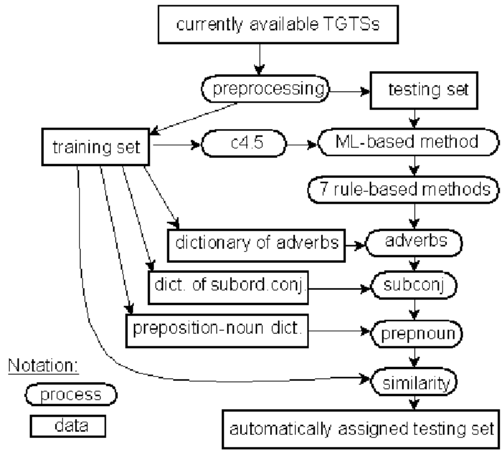


Figure 6.2: The architecture of the whole AFA system.

```
> foreachn.pl <hrany.fsp VstupniData/Test/*.fs   | cestinaOff |
    ml80.pl | pred.pl | verbs_active.pl |  verbs_passive.pl |
    pnom.pl | adjectives.pl | numerals.pl | pronounposs.pl |
    adverbs.pl | prepnoun.pl  > test_result.txt
```

And this is a segment from the resulting text file (only a few last columns are shown):

```
... reakce      reakce      nfs1a  n  1         sb    act   act   &verbs_active
... napr_i_klad napr_i_klad db     d  0         auxy  rhem
... lon_ske_m   lon_sky_    ais61a a  0         atr   rstr  rstr  &adjectives
... roce        rok         nis6a  n  6  v      adv   twhen
... dvana_cti   dvana_ct12  cbp2   c  2  mi_sto exd   rstr  rstr  &numerals
... zaplatili   zaplatit    vrmpa  v  0         pred  pred  pred  &pred
```

The architecture of the whole AFA system is shown in Figure 6.2. It depicts how the available data—after being split into the training and testing set—go through the system. The training set is used for the extraction of the dictionaries and for machine learning (C4.5). The testing set then goes through the sequence of modules (assigners) in which the functors are automatically assigned.

Taking advantage of the pipeline-fashioned execution of the assigners, I could examine many different permutations of the assigners—as it is discussed in the following chapter—without any additional effort.

For the automatic pre-annotation, the interface `foreachn.pl` is used. In the following example, the complete AFA system is applied on the file `bcb21trz.fs` and the functors are assigned in it:

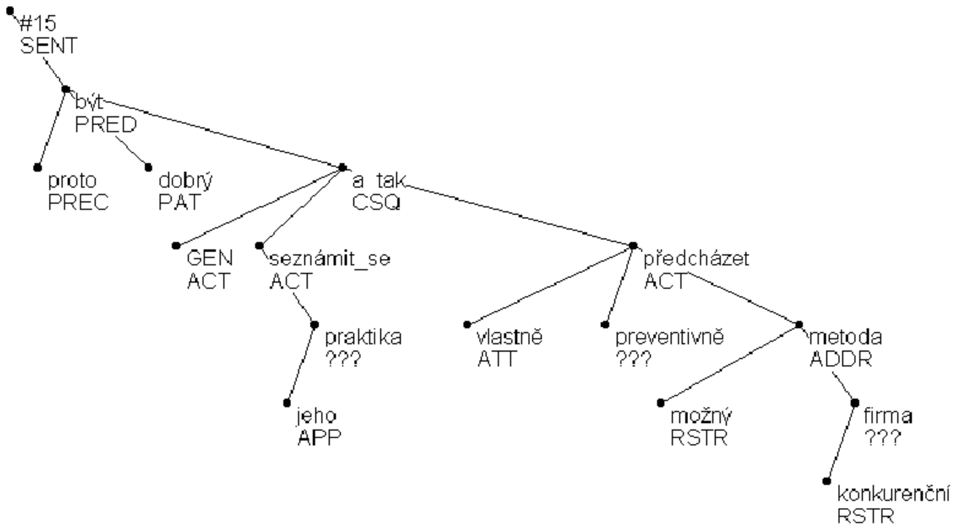


Figure 6.3: Tectogrammatical tree with automatically assigned functors.

```
> foreachn.pl <afa.fsp bcb21trz.fs
```

In the Perl program `afa.fsp`, a text line with 15 columns is generated for each node and sent into the pipeline of assigners. Then the 16th column is isolated (`cut -f16`) and the resulting functor is assigned to the appropriate attribute of the node. An example of an automatically annotated tectogrammatical tree is in Figure 6.3.

6.7 Further extensions

Since the presented AFA system has a very transparent architecture, it remains open for future improvements and extensions. The only condition for a new assigner is that it must fulfill the modularity requirements formulated at the beginning of section 6.2. Then it can be easily inserted into the pipeline of assigners, either directly in the command line (for testing purposes) or in the file `afa.fsp` (for the direct automatic annotation of a fs-file).

Chapter 7

Experiments and Results

7.1 How to measure AFA's performance

With respect to the "quality" of the individual methods of the AFA system, instead of being directly comparable (i.e., lying along one dimension), the methods should be rather placed into a two-dimensional space. The first coordinate corresponds to *precision* (it grows with minimization of the number of errors) and the other reflects *recall* (it grows with maximization of the number of correctly assigned functors). As it will be shown later, these two properties tend to be in opposition.[1]

In order to have a complete view on the AFA's qualitative characteristics, I measured several quantities for each assigner:

- *Cover* = the number of all nodes assigned by the given method
- *Relative cover* = cover divided by number of all functors to be assigned (1089 in the training set). This number also reflects the frequency of particular phenomenona (e.g., occurrences of possessive pronouns).
- *Errors* = the number of incorrectly assigned functors
- *Hits* = the number of correctly assigned functors
- *Recall* = the percentage of correct functor assignments by the given method among all the functors to be assigned (hit/1089·100%)
- *Precision* = the percentage of correct functor assignments by the given method among all functors assigned by this method (hits/cover·100%)

[1] Distinguisting between precision and recall is the standard way to describe the results which are —because of the complexity of the problem or imperfection of the solution—both incomplete and inconsistent, e.g., in [Baldwin–97].

All the measurements of the qualitative characteristics of AFA's components were evaluated exclusively using the tool `stat.pl` which is joined to the end of the pipeline of assigners. For example, after execution the command line

```
> foreachn.pl <hrany.fsp VstupniData/Test/*.fs | ml80.pl |
 pred.pl | verbs_active.pl | verbs_passive.pl | pnom.pl |
 adjectives.pl | numerals.pl | pronounposs.pl |  adverbs.pl |
 prepnoun.pl  | stat.pl
```

we obtain the following evaluation:

Number of lines: 1089 (100%)

Method	#Cover	#Hits	#Errors	Precision
ml80	406 (37.28 %)	384 (35.26 %)	22 (2.02 %)	94.58 %
adjectives	175 (16.06 %)	170 (15.61 %)	5 (0.45 %)	97.14 %
pronounpos	16 (1.46 %)	13 (1.19 %)	3 (0.27 %)	81.25 %
prepnoun	8 (0.73 %)	8 (0.73 %)	0 (0 %)	100 %
numerals	19 (1.74 %)	13 (1.19 %)	6 (0.55 %)	68.42 %
adverbs	28 (2.57 %)	24 (2.20 %)	4 (0.36 %)	85.71 %
pred	4 (0.36 %)	4 (0.36 %)	0 (0 %)	100 %
verbs_passive	7 (0.64 %)	6 (0.55 %)	1 (0.09 %)	85.71 %
verbs_active	21 (1.92 %)	18 (1.65 %)	3 (0.27 %)	85.71 %
	684 (62.80 %)	640 (58.76 %)	44 (4.04 %)	93.56 %

The first column contains the names of the assigners. In the second, third, and fourth columns, the numbers of occurrences are followed by the percentages in brackets; each percentage is expressed with respect to the number of all functors to be assigned, i.e., to the number of lines in the measured file. Obviously, these percentages are related to a different base than in the case of the precision calculation.

The evaluating script `stat.pl` is the exclusive source of the data discussed in this chapter.

7.2 Evaluation of experiments

Applying all the assigners available in the AFA system needn't necessarily be the most suitable solution for the purpose of automatic preprocessing during the transition from analytic to tectogrammatical structures in the PDT project, since the overall precision can be inacceptably low. Therefore, I needed to decide which assigners should be incorporated in the final AFA system.

The second question is in what order the assigners should be executed. When the covers (the sets of assigned functors) of individual assigners partly overlap each other, the assigners with the higher precision should be applied first. In such case, the order can play a very important role for the combined precision.

In order to be able to compose the optimal AFA configuration, I performed several measurements on different sequences of assigners. The results are in Tables 7.1-7.8, in which the assigners are presented in the same order in which they were executed. In each table, the quantitative characteristics described in the previous section are evaluated for each assigner separately as well as for the whole sequence of assigners. Let me remind, that the size of the training set is 6049 vectors and the size of the testing set is 1089 vectors.

The following measurements have been performed:

- Only the rule-based methods (RBMs) were applied on the testing set (Table 7.1); rel.cover=51.2%, prec.=93.9%.
- Since the RBMs are not directly dependent on the training set, they can and were applied also on the training set (Table 7.2); rel.cover=49 %, prec.=92.5%.
- Only the dictionary-based methods (DBMs) were applied on the testing set (Table 7.3); rel.cover=4.2%, prec.=89%.
- Both RBMs and DBMs were applied on the testing set (Table 7.4); rel.cover=55.5%, prec.=93.5%.
- Only the method `ml80` which is based on the machine learning was applied on the testing set (Table 7.5); rel.cover=37.2%, prec.=94.6%.
- Only the method `similarity` which is based on the nearest neigbour approach was applied on the testing set (Table 7.6); rel.cover=100%, prec.=73%.

- The RBMs, DBMs, and `ml 80` were applied on the testing set (Table 7.7); rel.cover=63%, prec.=93.4%.
- The `ml 80`, RBMs, and DBMs (Table 7.8) were applied on the testing set; rel.cover=63%, prec.=93.4%.
- All the available methods have been applied on the testing set in the order `ml80` , RBMs, DBMs, `similarity` (Table 7.9); rel.cover=100%, prec.=78.6%.

Method	Cover	Rel. cover	Hits	Recall	Errors	Precision
pred	104	9.6 %	104	9.6 %	0	100 %
verbs_active	199	18.3 %	184	16.9 %	15	92.5 %
verbs_passive	7	0.6 %	6	0.6 %	1	85.7 %
pnom	34	3.1 %	32	2.9 %	2	94.1 %
adjectives	177	16.2 %	170	15.6 %	7	96.0 %
numerals	21	1.9 %	15	1.4 %	6	71.4 %
pronounpos	16	1.5 %	13	1.2 %	3	81.3 %
Total	Σ 558	Σ 51.2 %	Σ 524	Σ 48.1 %	Σ 34	93.9 %

Table 7.1: Evaluation of the performance of the rule-based methods, when applied on the testing set.

In the remainder of this section I will point out a few facts that can be derived from the measured data.

The rule-based methods are not directly derived from the training set, that is why I could have applied them on the training set as well. So Tables 7.1 and 7.2 describe the performance of the same sequence of assigners on the two disjoint sets of data. The results achieved on the testing and training set are quite similar: relative recall is 51.2 % or 49 %, precision is 93.9 % or 92.5 %. This observation supports the conjecture that the performance of the rule-based method should not be drastically lower for any other PDT data.

Tables 7.7 and 7.8 show the performance of two sequences which contain the same assigners but in a different order (RBMs, DBMs and `ml80` versus `ml80`, RBMs, DBMs). The coverage of the respective families of methods is depicted in Figure 7.1. Surprisingly, the overall precision (93.4 %) and recall (63 %) of these

Method	Cover	Rel. cover	Hits	Recall	Errors	Precision
pred	574	9.5 %	554	9.2 %	20	96.5 %
verbs_active	973	16.1 %	907	15.0 %	66	93.2 %
verbs_passive	34	0.6 %	27	0.4 %	7	79.4 %
pnom	164	2.7 %	152	2.5 %	12	92.7 %
adjectives	1063	17.6 %	976	16.1 %	87	91.8 %
numerals	92	1.5 %	66	1.1 %	26	71.7 %
pronounpos	64	1.1 %	61	1.0 %	3	95.3 %
Total	Σ 2964	Σ 49.0 %	Σ 2743	Σ 45.3 %	Σ 221	92.5 %

Table 7.2: Evaluation of the performance of the rule-based methods, when applied on the training set.

Method	Cover	Rel. cover	Hits	Recall	Errors	Precision
prepnoun	9	0.8 %	9	0.8 %	0	100 %
adverbs	34	3.1 %	30	2.8 %	4	88.2 %
subconj	3	0.3 %	2	0.2 %	1	66.7 %
Total	Σ 46	Σ 4.2 %	Σ 41	Σ 3.8 %	Σ 5	Σ 89.1 %

Table 7.3: Evaluation of the performance of the dictionary-based methods, when applied on the testing set.

two sequences do not differ. This implies that in the intersection of the coverage of `ml80` and RBMs the (hand-written) rules achieve the same performance as the method based on machine learning. It can be a coincidence, but it is more likely that if the system C4.5 discovers a rule which has the same premise as one of the hand-written rules, then they have the same resulting functor too.

On the basis of a comparison of tables 7.4 and 7.7 we can conclude that the contribution of machine learning approach to the overall recall is 7 %.

One more interesting observation comes from the comparison of tables 7.5 and 7.9. If we employ the nearest vector approach (`similarity`) alone first, and then add the rule-based, dictionary-based and ML-based approaches, the improvement of precision is only 5.6 % (recall does not change, it is 100 % in both cases). This

Method	Cover	Rel. cover	Hits	Recall	Errors	Precision
pred	104	9.6 %	104	9.6 %	0	100 %
verbs_active	199	18.3 %	184	16.9 %	15	92.5 %
verbs_passive	7	0.6 %	6	0.6 %	1	85.7 %
pnom	34	3.1 %	32	2.9 %	2	94.1 %
adjectives	177	16.3 %	170	15.6 %	7	96.0 %
numerals	21	1.9 %	15	1.4 %	6	71.4 %
pronounpos	16	1.5 %	13	1.2 %	3	81.3 %
prepnoun	9	0.8 %	9	0.8 %	0	100 %
adverbs	34	3.1 %	30	2.8 %	4	88.2 %
subconj	3	0.3 %	2	0.2 %	1	66.7 %
Total	Σ 604	Σ 55.5 %	Σ 565	Σ 51.9 %	Σ 39	Σ 93.6 %

Table 7.4: Evaluation of the performance of the rule-based and dictionary-based methods, when applied on the testing set.

Method	Cover	Rel. cover	Hits	Recall	Errors	Precision
ml80	406	37.3 %	384	35.3 %	22	94.6 %
Total	Σ 406	Σ 37.3 %	Σ 384	Σ 35.3 %	Σ 22	Σ 94.6 %

Table 7.5: Evaluation of the performance of `ml80` (the method based on machine learning), when applied on the testing set.

Method	Cover	Rel. cover	Hits	Recall	Errors	Precision
similarity	1089	100 %	796	73.0 %	293	73.0 %
Total	Σ 1089	100 %	Σ 796	Σ 73.0 %	Σ 293	Σ 73.0 %

Table 7.6: Evaluation of the performance of `similarity` (the method based on the nearest vector approach), when applied on the testing set.

shows that the weights in the implementation of `similarity` were tuned well. But in contrast to the single method with 100 % coverage, the existence of the spectrum of methods enables to choose a compromise between precision and recall,

Method	Cover	Rel. cover	Hits	Recall	Errors	Precision
pred	104	9.6 %	104	9.6 %	0	100 %
verbs_active	199	18.3 %	184	16.9 %	15	92.5 %
verbs_passive	7	0.6 %	6	0.6 %	1	85.8 %
pnom	34	3.1 %	32	2.9 %	2	94.1 %
adjectives	177	16.3 %	170	15.6 %	7	96.0 %
numerals	21	1.9 %	15	1.4 %	6	71.4 %
pronounpos	16	1.5 %	13	1.2 %	3	81.3 %
prepnoun	9	0.8 %	9	0.8 %	0	100 %
adverbs	34	3.1 %	30	2.8 %	4	88.2 %
subconj	3	0.3 %	2	0.2 %	1	66.7 %
ml80	82	7.5 %	76	7.0 %	6	92.7 %
Total	Σ 686	Σ 63.0 %	$\Sigma\Sigma$ 641	Σ 58.9 %	Σ 45	Σ 93.4 %

Table 7.7: Evaluation of the performance of the sequence RBMs, DBMs, and `ml80`, when applied on the testing set.

Method	Cover	Rel. cover	Hits	Recall	Errors	Precision
ml80	406	37.3 %	384	35.3 %	22	94.6 %
pred	4	0.4 %	4	0.4 %	0	100 %
verbs_active	21	1.9 %	18	1.7 %	3	85.7 %
verbs_passive	7	0.6 %	6	0.6 %	1	85.7 %
adjectives	175	16.1 %	170	15.6 %	5	97.1 %
numerals	19	1.7 %	13	1.2 %	6	68.4 %
pronounpos	16	1.4 %	13	1.2 %	3	81.3 %
prepnoun	8	0.7 %	8	0.7 %	0	100 %
adverbs	28	2.6 %	24	2.2 %	4	85.7 %
subconj	2	0.2 %	1	0.1 %	1	50 %
Total	Σ 686	Σ 63.0 %	Σ 641	Σ 58.9 %	Σ 45	93.4 %

Table 7.8: Evaluation of the performance of the sequence `ml80`, RBMs, and DBMs, when applied on the testing set.

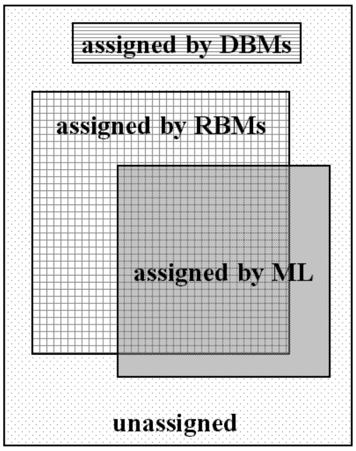


Figure 7.1: Comparison of the covers of individual families of methods for the sequence machine learning, rule-based methods, dictionary based methods. The outermost rectangle depicts the set of all functors to be assigned in the testing set.

as it will be shown in the next section.

RBMs and `ml80` ignore lexical attributes of the nodes (word form, lemma),

Method	Cover	Rel. cover	Hits	Recall	Errors	Precision
ml80	406	37.3 %	384	35.3 %	22	94.6 %
pred	4	0.4 %	4	0.4 %	0	100 %
verbs_active	21	1.9 %	18	1.7 %	3	85.7 %
verbs_passive	7	0.6 %	6	0.6 %	1	85.7 %
adjectives	175	16.0 %	170	15.6 %	5	97.1 %
numerals	19	1.7 %	13	1.2 %	6	68.4 %
pronounpos	16	1.5 %	13	1.2 %	3	81.3 %
prepnoun	8	0.7 %	8	0.7 %	0	100 %
adverbs	28	2.6 %	24	2.2 %	4	85.7 %
subconj	2	0.2 %	1	0.1 %	1	50 %
similarity	403	37.0 %	215	19.7 %	188	53.3 %
Total	Σ 1089	Σ 100 %	Σ 856	Σ 78.6 %	Σ 233	78.6 %

Table 7.9: Results of all the methods on the testing set.

the only exceptions are prepositions and subordinating conjunctions. From the Tables 7.3 and 7.7 it can be computed that the recall of the assigning sequence RBMs, `ml80` is 55 %. In other words, at least one half of functors can be assigned without the slightest idea about 'what the sentence is about'.

7.3 Precision versus recall

As I already mentioned, it is possible to select and apply only a subset of the available methods and thus control the characteristics of the AFA system. It should be decided whether to prefer to *minimize the number or errors*, thus maximazing precision, or *maximize the number of correctly assigned nodes*, thus maximizing recall. This choice is very explicit. The optimal compromise should be influenced by the *misclassification cost* corresponding to the amount of annotators' work involved in finding and correcting a wrongly assigned functor. However, estimating the misclassification cost would require additional experiments with the annotators, in order to perform the necessary measurements of annotators' performance. This would in turn imply an additional load for them, which is in contradiction with the main goal of this thesis (decreasing the amount of annotators' work).

The relation between recall and precision is depicted in Figure 7.2. The highest recall is achieved when all methods are applied. Unfortunately, the overall precision 78.6 % is not acceptable, since the resulting automatically annotated files would require too many manual corrections. Precision grows to an acceptable level if the method `similarity` is removed (precision 93.4 %, recall 58.9 %). Therefore, I think that the most feasible compromise between precision and recall is the sequence `ml80`, RBMs, DBMs.

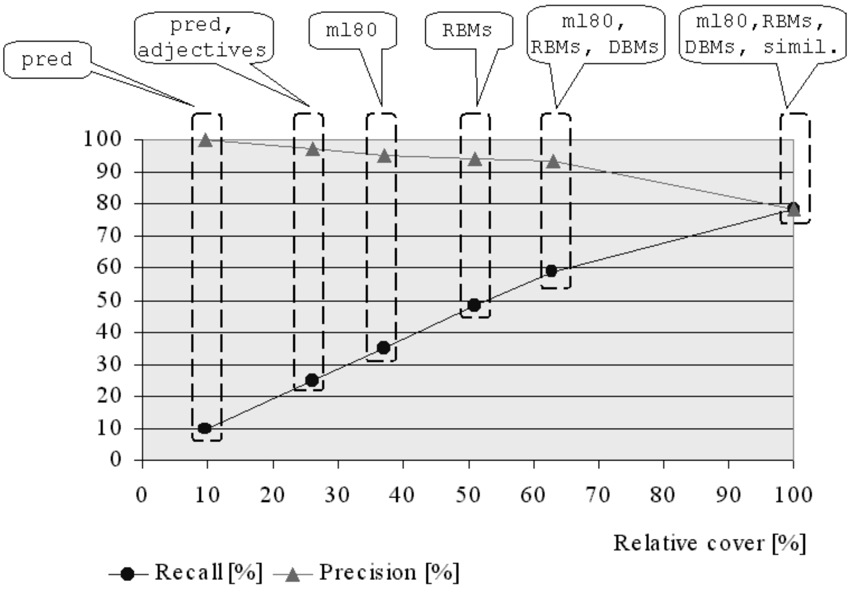


Figure 7.2: Precision versus Recall. This picture depicts the performance of selected sequences of assigners. Obviously, the higher the recall achieved, the lower the precision.

Chapter 8

Conclusions

Die Umgangssprache ist ein Teil des menschlichen Organismus und nicht weniger kompliziert als dieser.

Ludwig Wittgenstein

The goal of this thesis. The goal of this thesis was to design, implement and evaluate a system for automatic functor assignment within the Prague Dependency Treebank projekt at the Institute of Formal and Applied Linguistics. Such a tool should reduce the manual annotation effort during the transition from the analytical tree structures to the tectogrammatical tree structures, which otherwise consumes a huge amount of time of linguistic experts.

The contribution of this thesis. The presented AFA system is based on the hypothesis that when a functor is to be assigned to a node, then in a significant subset of the cases sufficient information for this decision can be acquired from the node itself and from the parent node. Using this assumption, I constructed a system that profits from the symbiosis of different approaches, namely rule-based methods, dictionary-based methods, machine learning approach, and nearest vector approach. During the development of the AFA system, I used the available manually annotated tectogrammatical tree structures for training and testing purposes.

The overall performance (recall versus precision) of the resulting AFA system can be tuned by combining selected methods in various ways. Either all functors are assigned and the precision is 78.6 %, or 63.0 % of nodes are assigned with the precision 93.4 %. The implementation of the latter approach is ready to be used at the IFAL since September 2000. No other AFA system with comparable recall was available before.

Discussion. Since I had only very limited testing set, the question about the reliability and extensibility of the achieved results naturally arises. When the system is used on new data, the performance can be expected to decrease for two reasons. Firstly, I tested the AFA system on tectogrammatical trees which were not only manually annotated with functors, however, their topology was manually revised. The topology of the new tectogrammatical tree structures is generated automatically from the analytical tree structures. If this procedure generates also some topological mistakes in the trees, then these mistakes will inevitably influence the performance of the AFA system. Secondly, a part of the involved dictionaries was mined from the training data. If the new trees to be assigned represent sentences with very distant topic and genre, then the recall of the dictionary-based methods is likely to decrease, since "new" words (those not observed in the training set) will appear.

Obviously, the real contribution of the presented system, i.e., its usefulness for the annotators, can only be evaluated after a period of its use in the actual annotation process.

Future work. The potential of the AFA was undoubtedly not fully exploited in this thesis. But the future improvements of the AFA system, which will increase the recall while keeping the precision high, will probably require extensive utilization of linguistic resources which are not available yet (e.g., tectogrammatically annotated lexicon of verb valency frames) and a larger and more diverse training set of the PDT data. However, one can hardly expect a system that would be able to completely substitute the experts for the tectogrammatical annotation. At least not in the near future.

Bibliography

[AA-91] Aijmer, K., Altenberg, B. (eds.): *English Corpus Linguistics.* Longman, New York, 1991.

[Allen–95] Allen, J.: *Natural Language Underestanding.* The Benjamin/Cummings Publishing Company, 1995.

[Baldwin–97] Baldwin, B.: *CogNIAC: High Precision Coreference with Limited Knowledge and Linguistic Resources.* Operational Factors in Practical Robust Anaphora Resolution for Unrestricted Texts, Proceedings of the Workshop at the Annual Meeting of the ACL'97, Universidad Nacional de Educación a Distanci, Madrid, pp. 38-45, 1997.

[BPS–99] Böhmová, A., Panevová, J., Sgall, P.: *Syntactic Tagging: Procedure for the Transition from the Analytic to the Tectogrammatical Tree Structures.* Text, Speech and Dialogue, Springer, pp. 34–38, 1999.

[BH–99] Böhmová, A., Hajičová, E.: *How much of the underlying syntactic structure can be tagged automatically?* In: The Prague Bulletin of Mathematical Linguistics 71, Charles University, Prague, 1999.

[Čermák–99] Čermák, F.: *Information, Language, Corpus and Linguistics.* Text, Speech and Dialogue, Springer, pp. 39–43, 1999.

[DePryck–93] DePryck, K.: *Knowledge, Evolution and Paradox. The Ontology of Language.* State University of New York Press, 1993.

[Erjavec-Ide–98] Erjavec, T., Ide, N.: *The MULTEXT-East Corpus.* First International Conference on Language Resources and Evaluation, 1998.

[Hajič–98] Hajič, J.: *Building a Syntactically Annotated Corpus: The Prague Dependency Treebank.* Issues of Valency and Meaning, Karolinum, 1998.

[Hoffman–95] Hoffman, B.: *Computational Analysis of the Syntax and Interpretationf of 'Free' Word-order in Turkish.* Ph.D. thesis, University of Pennsylvania, 1995.

[Jansen–96] Jansen, T.M.V.: *Composionality.* Handbook of Logic and Language. Elsevier Science, pp. 419-475, 1996.

[Kazakov–99] Kazakov, D.: *Natural Language Processing Applications of Machine Learning.* Ph.D. thesis, Department of Cybernetics, Czech Technical University, 1999.

[Kotek et al.–80] Kotek, Z., Chalupa, V., Bruha, I., Jelínek, J.: *Adaptivní a učící se systémy.* SNTL, 1980.

[Kruijff–98] Kruijff, G. M.: *Basic Dependency-based Logical Grammar.* Technical Report ÚFAL-TR-5, 1998.

[Kruijff–99] Kruijff, G. M.: *Dependency Grammar Logic. An Introduction*, manuscript available at http://cogsci.ed.ac.uk/ gj , 1999.

[Kruijff–01] Kruijff, G. M.: *A Categorial-Modal Architecture of Informativity.* Ph.D. thesis, Faculty of Mathematics and Physics, Charles University, Prague, 2001 (to appear).

[KK–99] Kruijff, G., Kruijff-Korbayová, I.: *Text structuring in a multilingual system for generation of instructions.* Text, Speech and Dialogue, Springer, pp. 89–94, 1999.

[Křen-96] Křen, M.: *Editor grafů.* Master thesis, Faculty of Mathematics and Physics, Charles University, Prague, 1996.

[Kučera–99] Kučera, K.: *The General Principles of the Diachronic Part of the Czech National Corpus.* Text, Speech and Dialogue, Springer, pp. 62–65, 1999.

[Lager–95] Lager, T.: *A Logical Approach to Computational Corpus Linguistics.* Ph.D. thesis, Department of Linguistics, Göteborg University, 1995.

[Manual1] Bémová, A., et al: *Anotace na analytické rovině: návod pro anotátory.* Technical Report ÚFAL-TR-4, 1997.

[Manual2] Hajičová, E., Panevová, J., Sgall, P.: *Manuál pro tektogramatické značkování.* Technical Report ÚFAL-TR-7, 1999.

[Melichar–97] Melichar, B.: *Languages and Translations.* Czech Technical University, Prague, 1997.

[MR–91] Muller, B., Reinhardt, J.: *Neural Networks – An Introduction.* Springer, 1991.

[Pala–99] Pala, K.: *Semantic Annotation of (Czech) Corpus Texts.* Text, Speech and Dialogue, Springer, pp. 56–61, 1999.

[Panev–80] Panevová, J.: *Formy a funkce ve stavbě české věty.* Academia, Prague, 1980.

[Pere–98] Peregrin, J.: *Obrat k jazyku: druhé kolo.* FILOSOFIA, Prague, 1998.

[SHP–86] Petr Sgall, Eva Hajičová, and Jarmila Panevová: *The Meaning of the Sentence in its Semantic and Pragmatic Aspects.* Reidel, Dordrecht, The Netherlands, 1986.

[LD–94] Lavrač, N., Džeroski, S.: *Inductive Logic Programming.* Techniques and Applications, Chichester, UK, 1994.

[Steedman-98] Steedman, M.: *Categorial Grammar.* The MIT Encyclopedia of Cognitive Sciences, MIT Press, 1998.

[Strossa–99] Strossa, P.: *Vybrané kapitoly z počítačového zpracování přirozeného jazyka.* Silesian University, Institute of Informatics, Opava, 1999.

[Šulc–99] Šulc, M.: *Korpusová lingvistika.* Karolinum, Prague, 1999.

[RL–95] De Raedt, L., Lavrač, N.: *Introduction to Inductive Machine Learning.* Lecture notes for a course in machine learning, J. Stefan Institute, Ljubljana, 1996.

[Will–93] Willems, M.: *Chemistry of Language.* Ph.D. thesis, Department of Applied Mathematics, University of Twente, 1993.

[Witt–22] Wittgenstein, L.: *Tractatus logico-philosophicus.* Institut pro středoevropskou kulturu a politiku, Prague, 1993.

[ZŽ–00] Žabokrtský, Z.: *Automatic Functor Assignment in the Prague Dependency Treebank.* Text, Speech and Dialogue, Springer, pp. 45–50, 2000.

Appendix A

Armchair linguistics vs. corpus linguistics

Armchair linguistics does not have a good name in some linguistics circles. A caricature of the armchair linguist is something like this. He sits in a deep soft comfortable armchair, with his eyes closed and his hands clasped behind his head. Once in a while he opens his eyes, sits up abruptly shouting, "Wow, what a neat fact!", grabs his pencil, and writes something down. Then he paces around for a few hours in the excitement of having come still closer to knowing what language is really like. (There isn't anybody exactly like this, but there are some approximations.)

Corpus linguistics does not have a good name in some linguistics circles. A caricature of the corpus linguist is something like this. He has all of the primary facts that he needs, in the form of approximately one zillion running words, and he sees his job as that of deriving secondary facts from his primary facts. At the moment he is busy determining the relative frequencies of the eleven parts of speech as the first word of a sentence versus as the second word of a sentence. (There isn't anybody exactly like this, but there are some approximations.) These two don't speak to each other very often, but when they do, the corpus linguist says to the armchair linguist. "Why should I think that what you tell me is true?", and the armchair linguist says to the corpus linguist, "Why should I think that what you tell me is interesting?"

Charles Fillmoore (1992)

Appendix B

List of Functors

All the following examples are authentic, they occured in the training set, and *vice versa*, the functors which did not appear in the training set at all, are not listed.

ACMP	**Se žádostmi** o výjimku je nutné se obrátit na radu města.
ACT	Moje **firma** vyrobila na zakázku zboží pro zákazníka ...
ADDR	V Plzni je **stánkařům** k dispozici tržnice ...
ADVS	... do ceny bytů se promítne řada faktorů, zejména však amortizace.
AIM	Hospoda byla jen startem, polem **k podnikání** s masem a masnými výrobky.
APP	Provoz má přece už **svůj** rytmus.
APPS	... však neřeší základní problém, **a to** volné, bezbariérové průchodnosti ...
ATT	**Samozřejmě** existují počítačové programy, které využíváme ...
BEN	Profit připravuje pro své **čtenáře** poradnu pro diky.
CAUS	Věděl díky letité praxi, že obyvatelé z okolních domů ...
CNCS	Od něj získal vnuk výtečné základy, ač sám vystudoval školu zaměřenou na dopravu.
COMPL	Jako hlavní **zlo** vidím velké množství daní ...
COND	Když o někom **řekneme**, že je zloděj ...
CONJ	V Prazei v jiných velkých městech je pochůzkový **a** stolkový prodej na ulicích zakázaný.
CPR	Pokud budeme postupovat stejnou metodikou, jako **je propočten** fond pracovní doby v Německu ...
CRIT	Podle předběžných **odhadů** se totiž počítá ...

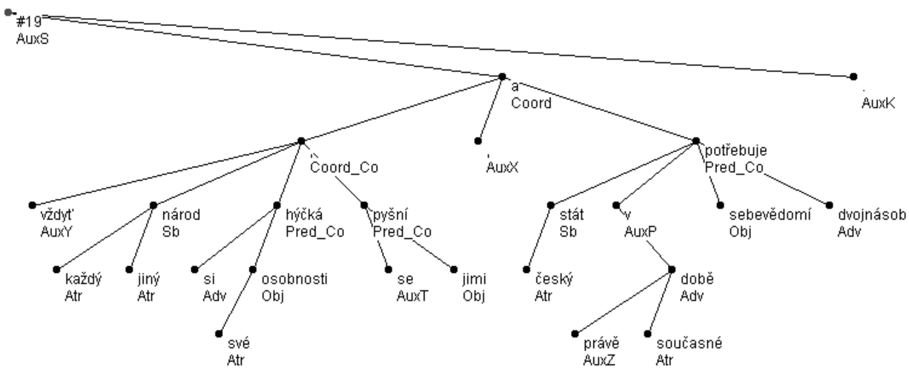
CSQ	... vhodná pozornost dokáže vytvořit prostředí důvěry a sympatie, takže určité ledy a bariéry rezervovanosti se brzy **rozplynou**.
DENOM	**Šance** pro movité nájemníky.
DIFF	Současná daňová soustava funguje o **něco** více než rok.
DIR1	Na začátku je nejdůležitější ujasnit si cíle a pak **z cesty** neustupovat.
DIR2	... jako když se prodíráte **křovím** a v dáli svítí mýtina.
DIR3	Podnikatel má sledovat vývoj ve svém oboru a doslova táhnout svoji firmu **dopředu** ...
EFF	Přitom jen **za materiál** pro uvedenou zakázku jsme vynaložili přes 150 tisíc korun.
EXT	**Celkem** zaměstnávám zhruba stovku lidí.
ID	Je tu pro vás připravena rubrika Daňový **poradce**.
INTF	Uvědomuje se, že u nás by **to** nešlo ...
INTT	A kuchař, který vynikající pokrmy připraví, se přijde za uznání hostů **poděkovat**.
LOC	**V Plzni** je stánkařům k dispozici tržnice ...
MANN	**Klidně** jsem mohl seskočit a dál dělat ve státním podniku, nic by se nestalo.
MAT	Firma produkuje na padesát sortimentních druhů **párků**, ...
MEANS	Nedat na první dojem, **jakým** na nás zákazník působí ...
MOD	Podnikání je **bezpochyby** krutá dřina, ale krásná.
NORM	Snad na základě **reklamy**, i když se zdá, že tentokrát ...
ORIG	... nenápadný človíček, **z něhož** se může vyklubat špión ...
PAR	Začal jsem, **řekněme**, jako provazochodec.
PAT	Napsali jsme novou **urgenci**.
PREC	Myslel jsem si **totiž**, že už všechno umím.
PRED	**Zabývám** se mezinárodní kamiónovou přepravou.
REG	Drobnější podniky se také účelově sdružují u větších zakázek.
RESL	Policie tak jen **bezmocně** přihlíží ...
RESTR	Naše platné právo kromě trestněprávní **odpovědností** umožňuje postihnout nelegální metody ...
RHEM	Stále **ještě** mohou lidé začít.
RSTR	**Kvalitní** boty dnes stojí dvakrát i čtyřikrát více ...

SUBS	Místo vlastního **rozhodování** o svých akciích ...
TFHL	Obuv na víc než jednu sezónu vyžaduje péči i opravy.
TFRWH	Původní rozhodnutí vlády odročeno z **1.4.** na 1.5.
THL	Dělal jsem bez přestávky celé **týdny**, často v noci.
THO	Křoví je husté a **často** neprostupné.
TOWH	Původní rozhodnutí vlády odročeno z 1.4. na **1.5.**
TPAR	Při letošním **udílení** ceny Grammy byla ...
TSIN	Od té **doby** uplynulo už několik měsíců, ...
TTILL	Na Vaše dotazy, které nám zašlete do redakce do 5. **dubna** ...
TWHEN	Můžeme je prodávat i **letos**.

Appendix C

Examples of the analytical and tectogrammatical tree structures

(a) Analytical tree structure



(b) Tectogrammatical tree structure

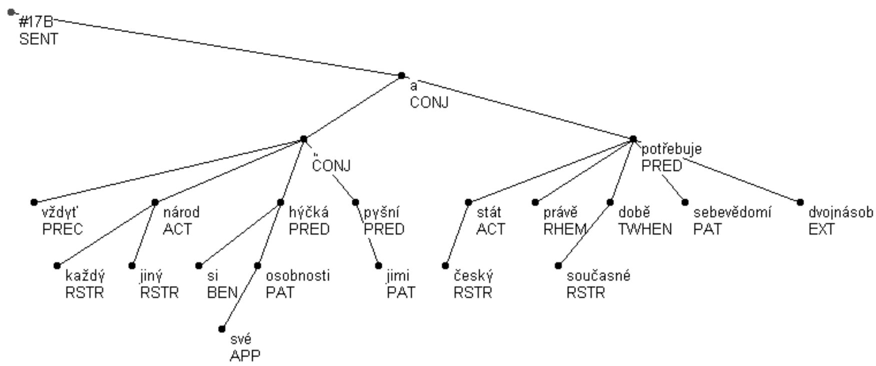
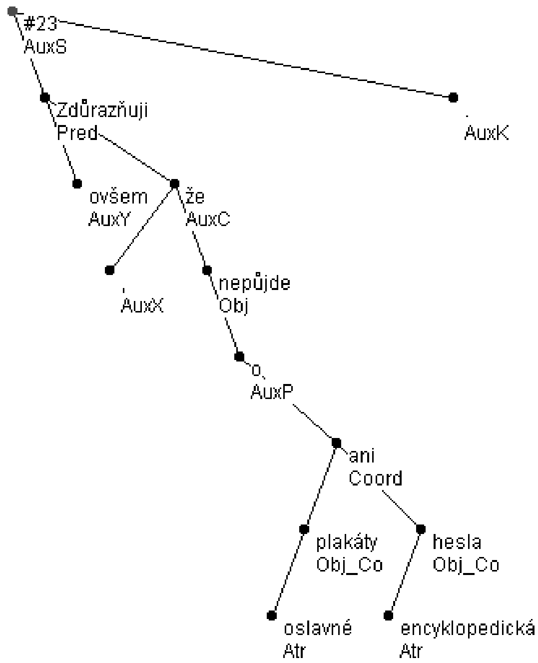


Figure C.1: Analytical and tectogrammatical tree structures of the sentence "*Vždyť každý jiný národ si své osobnosti hýčká, pyšní se jimi, a český stát právě v současné době potřebuje sebevědomí dvojnásob.* "

(a) Analytical tree structure



(b) Tectogrammatical tree structure

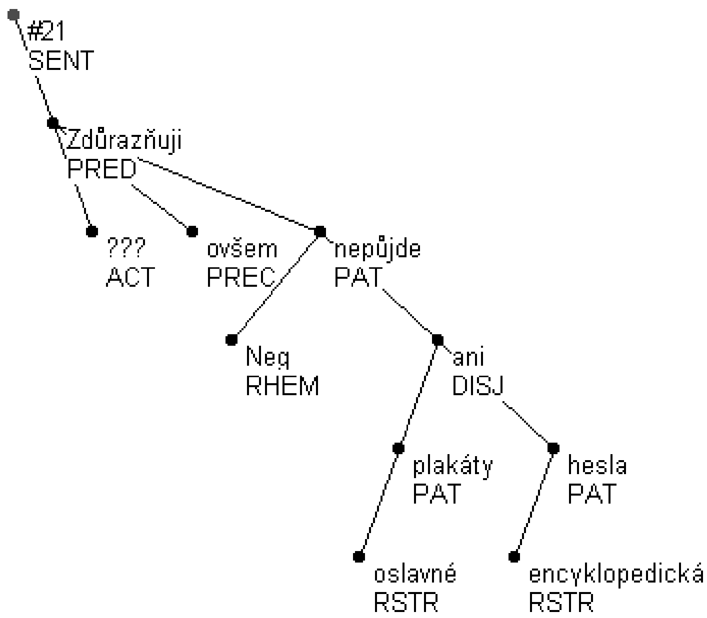
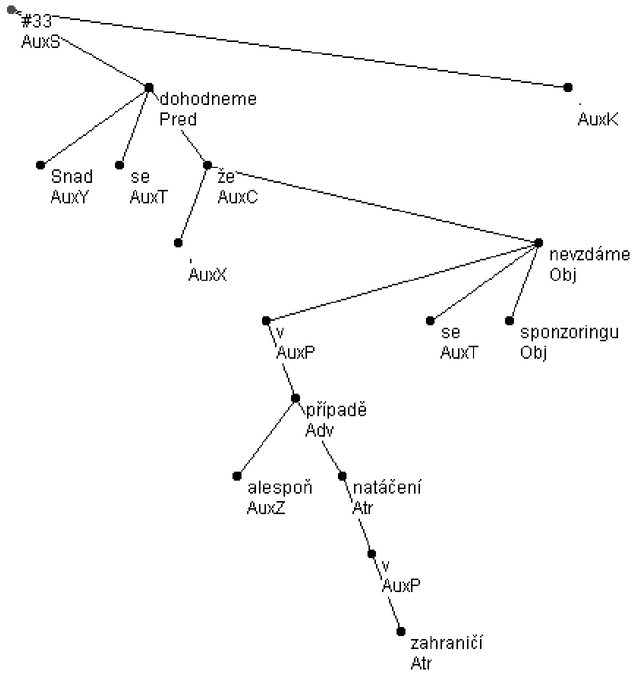


Figure C.2: Analytical and tectogrammatical tree structures of the sentence "*Zdůrazňuji ovšem, že nepůjde o slavné plakáty ani encyklopedická hesla.* "

(a) Analytical tree structure



(b) Tectogrammatical tree structure

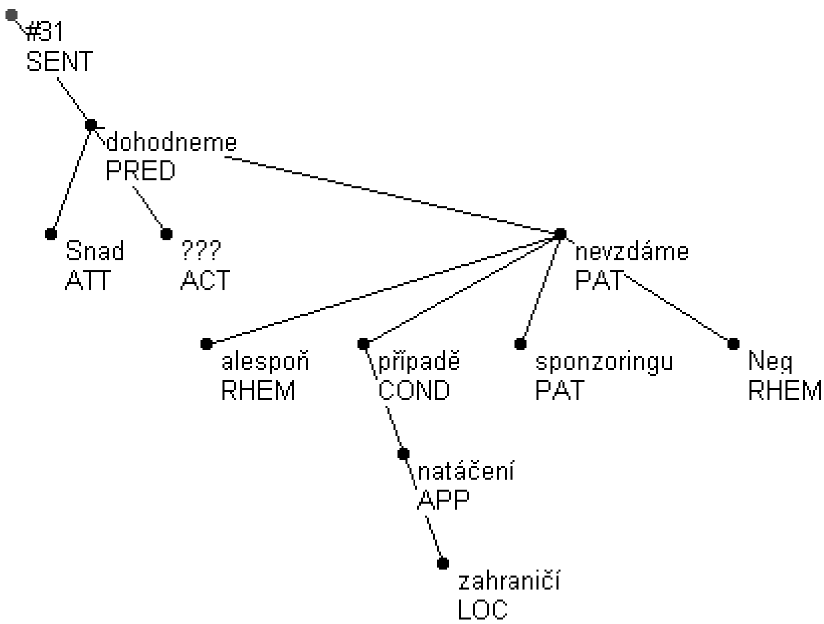
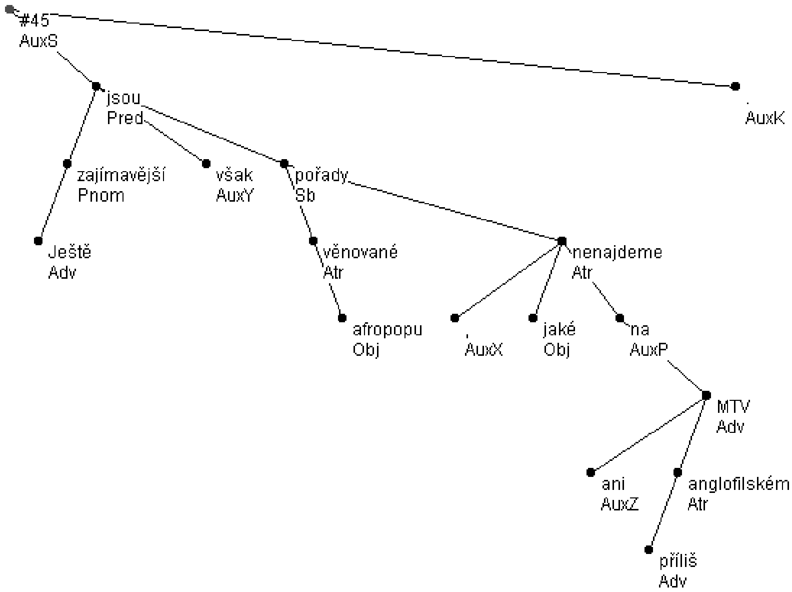


Figure C.3: Analytical and tectogrammatical tree structures of the sentence "*Snad se dohodneme, že alespoň v případě natáčení v zahraničí se sponzoringu nevzdáme.*"

(a) Analytical tree structure



(b) Tectogrammatical tree structure

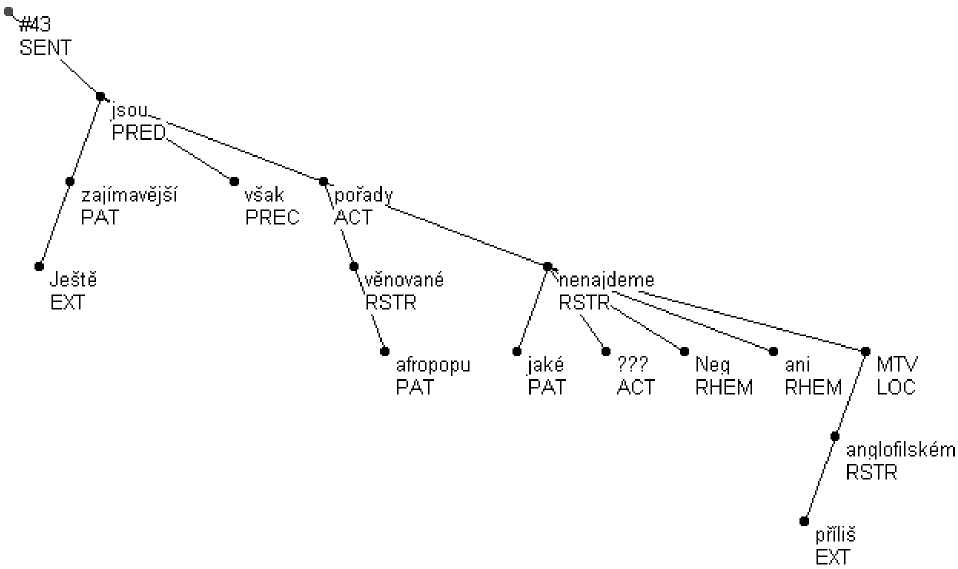


Figure C.4: Analytical and tectogrammatical tree structures of the sentence "*Ještě zajímavější jsou však pořady věnované afropopu, jaké nenajdeme ani na příliš anglofilském MTV.*"

Appendix D

Equivalence classes of verbs with respect to their valency

The following list contains the ten largest equivalence classes that were induced by an equality relation defined on their valency frames, as described on page 49.

1. Class No. 1 (271 verbs), #4 : aktivovat, aktualizovat, bodovat, deformovat, dokončit, hroutit, ignorovat, instalovat, kvasit, mobilizovat, monitorovat, obdivovat, odpracovat, odsouhlasit, ohlédnout, okrást, okupovat, pochopit, podstupovat, pozměnit, projektovat, prostudovat, prozkoumat, prošetřit, předpokládat, tradovat, utlumit, varovat, vychutnávat, vyřknout, zapříčinit, zdražovat, znásilnit, zpochybnit, zpronevěřit, zvedat, zvládnout, ...

2. Class No. 2 (245 verbs), #4 #7 : ctít, dotovat, klestit, mást, mýlit, narovnat, obtěžovat, odemknout, oslabovat, ovládat, pojmenovat, rozzlobit, uchvátit, ujistit, ukončit, uživit, vítat, zakrývat, zaplňovat, zesílit, zobrazit, ...

3. **Class No. 3 (138 verbs), #4 #3 :** doručit, nabízet, nelhat, odejmout, odepřít, odpustit, podřizovat, prezentovat, prodlužovat, projevit, předpovídat, přislíbit, přisoudit, režírovat, sdělovat, snižovat, snížit, vypršet, vytknout, zamítnout, zdanit, zpřístupnit, šéfovat, ...

4. **Class No. 4 (78 verbs), se #7 :** budit se, doplnit se, líčit se, nalít se, namalovat se, obhajovat se, oživit se, oživovat se, pobouřit se, polít se, pominout se, prodloužit se, přičinit se, spasit se, trávit se, ujít se, unášet se, utvářet se, uživit se, vzrušit se, zabezpečovat se, zaplnit se, zaplňovat se, zasloužit se, zastřelit se, znepokojit se, znepokojovat se, zvětšit se, ...

5. **Class No. 5 (64 verbs), #4 #7 v_6 :** charakterizovat, hrabat, korigovat, napodobit, napodobovat, obohacovat, obohatit, opomíjet, ovlivňovat, ověřit,

podepřít, podvádět, poškodit, poškozovat, prověřit, prověřovat, předhánět, předstihnout, překonat, překonávat, překvapit, překvapovat, převýšit, sílit, ubezpečit, upřednostňovat, zabrzdit, zabydlet, zdokonalovat, zhoršit, zjednodušit, zjednodušovat, ...

6. **Class No. 6 (64 verbs), #4 #7 #3 :** dosvědčit, garantovat, kompenzovat, komplikovat, nahradit, nahrazovat, odůvodňovat, oplatit, podepisovat, potvrdit, potvrzovat, protrhnout, překazit, rozumět, třít, usnadňovat, vyjadřovat, vylepšit, vylepšovat, vylíčit, vyslovit, zabezpečit, zmařit, změřit, zničit, způsobit, způsobovat, ztěžovat, ztížit, ...

7. **Class No. 7 (58 verbs), si #4 :** chválit si, chytnout si, domyslit si, klást si, nadělat si, obejít si, plánovat si, pokazit si, popudit si, prohlédnout si, předejít si, přemoci si, připisovat si, rozdat si, rozmyslit si, slíbit si, ujasnit si, vypít si, vyslechnout si, vysvětlit si, vytknout si, vyčítat si, změřit si, říkávat si, ...

8. **Class No. 8 (53 verbs), se #7 v_6 :** konkretizovat se, otřást se, podepřít se, pohoršit se, prohloubit se, projevit se, předhánět se, přežívat se, rozhýbat se, rozmnožovat se, rozvíjet se, ujistit se, ujišťovat se, utvrdit se, uvést se, zahltit se, zastírat se, zavést se, zdokonalit se, zhoršovat se, zlepšit se, zmenšovat se, zmítat se, zpevnit se, zpomalit se, zrcadlit se, ztělesňovat se, ...

9. **Class No. 9 (51 verbs), se s_7 :** hrát se, hádat se, milovat se, měřit se, namáhat se, obejmout se, objímat se, oženit se, pohádat se, poprat se, poradit se, prohodit se, přitáhnout se, sbližovat se, sehrát se, seznamovat se, seznámit se, shledávat se, sloučit se, smiřovat se, smířit se, soudit se, střetávat se, tahat se, utěšovat se, vadit se, vodit se, vsadit se, vypořádat se, vítat se, zapomenout se, ztotožnit se, ženit se, ...

10. **Class No. 10 (48 verbs), #4 v_6 :** brzdit, hájit, koupat, navštívit, novelizovat, podporovat, podpořit, preferovat, prolomit, provozovat, přeceňovat, rozhodnout, rozpouštět, tolerovat, vyjmenovat, vylosovat, vytušit, věznit, zhasnout, zmiňovat, zmínit, ztvárnit, ztělesňovat, zužovat, zvýhodňovat, zúžit, ...