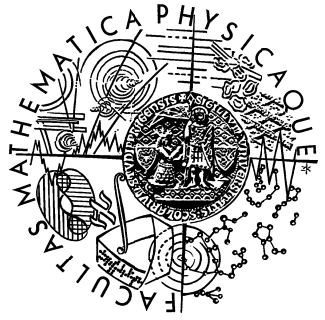
Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Analytical and Tectogrammatical Analysis of a Natural Language

Václav Klimeš



PhD thesis

Tutor: doc. RNDr. Jan Hajič, Dr.

Prague, 2006

Abstract

The thesis presents tools for analysis at analytical and tectogrammatical layers that the Prague Dependency Treebank is based on.

The tools for analytical annotation consist of two parsers and a tool for assigning syntactic tags. Although the performance of the parsers is far below that of the state-of-the-art parsers, they both can be considered a certain contribution to parsing, since the methods they are based on are novel. The tool for assigning syntactic tags makes 15% less errors than a tool used for this purpose previously.

The tool developed for tectogrammatical annotation is the only one that can currently perform this task in such a breadth. Although other, specialized tools may have a better performance of some of its particular subtasks, my tool makes 29% and 47% less errors for the Czech language than the combination of existing tools for annotating the tectogrammatical structure and deep functors, respectively, which are the core of the tectogrammatical layer.

The proposed tools are designed the way they can be used for other languages as well.

Resumé

Tato práce předkládá nástroje pro analýzu na analytické a tektogramatické rovině, které jsou základem Pražského závislostního korpusu.

Nástroje pro analytickou anotaci sestávají ze dvou parserů a nástroje přiřazujícího tzv. analytické funkce. Ačkoli úspěšnost parserů je daleko za úspěšností nejlepších parserů, oba mohou být chápány jako určitý přínos k parsingu, neboť jsou založeny na nových metodách. Nástroj přiřazující analytické funkce dělá o 15 % chyb méně než nástroj, který se k tomuto účelu používal dosud.

Nástroj vyvinutý pro tektogramatickou anotaci je jediný, který tuto úlohu nyní zvládá v takové šíři. Ačkoli jiné, specializované nástroje možná řeší některé její podúlohy lépe, pro češtinu dělá můj nástroj o 29 %, resp. 47 % méně chyb než kombinace existujících nástrojů určujících tektogramatickou strukturu, resp. hloubkové funktory, což je obojí jádrem tektogramatické roviny.

Předkládané nástroje jsou navrženy tak, aby je bylo možno použít i pro jiné jazyky.

Acknowledgements

My soul is filling with gratitude and humility when I realize how many people have contributed one way or another to this thesis.

First of all I thank my wife Šárka for her encouragement, support, care, and love.

My thanks go to my colleagues from Institute of Formal and Applied Linguistics and dozens of annotators as well, which I largely even do not know personally, not only for the data and the tools, which this thesis could not be written without, but also for friendly and supportive work environment they have created. Above all I want to thank Zdeněk Žabokrtský for his comments to my work and his inspiring ideas; Daniel Zeman who showed me with his own PhD thesis how such a thesis should look like; and Petr Pajas who was always prepared to trace both my and his own bugs ☺ appearing during my usage of his excellent *btred* tool, in which all the code of this work is programmed.

I thank my tutor and boss Jan Hajič for his effort in balancing my research with my other work duties and for his rare but peremptory advice regarding the direction of my research.

Outside the Institute I wish to thank Radu Florian from Johns Hopkins University, Baltimore, MD, for his *fnTBL* toolkit, which I used for the computations described in the thesis. I also thank to Tomáš Machalík for performing corrections of this text.

I thank several institutions which funded my research, namely for the following grants: The grant of the Grant Agency of the Czech Republic No. 405/03/0913, the project of the MŠMT ČR No. LN00A063 (*Center for Computational Linguistics*), and the *Information Society* project of the Czech Academy of Sciences No. 1ET10147016.

Although I have exploited help of the people mentioned above, the writing being submitted is my work, which also holds true for its deficiencies and mistakes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

'Where shall I begin, please, your Majesty?' he asked.
'Begin at the beginning,' the King said gravely,
'and go on till you come to the end: then stop.'
[Carroll, 2003]

1.1 Motivation

"The goal of the PhD thesis is to create an automatic analyzer creating tectogrammatical structure of a natural language. The analyzer would be based on quantitative methods, so that an annotated language corpus will be the only thing needed for its training. The analyzer will be independent of the type of future applications and of the language and domain of the training corpus; experiments will be performed on the Czech, English, and Arabic language. Experiments with several versions of language models will be performed as well. Objective criteria will be used for the evaluation of quality of the analyzer. The analyzer will be brought to the form of a complete software package."

Despite of the language, which is originally Czech, these are exactly the words describing the goal of my PhD thesis, as they stand in its submission. They give the topic of this thesis, however, they do not state why one should solve such a task.

There are several reasons for developing a tool for performing the annotation at the tectogrammatical layer, sometimes called "the layer of deep syntax". Annotation of a sentence at this layer is closer to the meaning of a sentence than its surface-syntactic annotation and this is why the information captured

at the tectogrammatical layer is crucial for machine understanding of a natural language. Hence, such a tool can be used for machine translation, information retrieval and human–computer communication; however, it can help in other tasks as well. Obviously, the tool can be employed by annotators creating tectogrammatical representation (which can be used for training tools performing the noticed tasks afterwards) to preannotate the data and thus to ease their work.

1.2 Theoretical background

As stated above, the goal of the thesis is to develop a tool performing tectogrammatical analysis, which uses just an annotated language corpus for obtaining its language model. Since there is only one corpus having human tectogrammatical annotation available, the choice of the "pilot" corpus is clear: to use the Prague Dependency Treebank, a collection of annotated Czech texts. Its annotation is based on the theory of language description called Functional Generative Description, shortly FGD, and the corpus is the proof of its viability. This theory develops from long-term tradition of so-called Pragian School dating back to 1926. FGD has been introduced in 1960s and summarized in [Sgall et al., 1986]. Two of its features are important for my work. The first one is the view of a language as of a system of more layers of language description: forms of units at a lower layer (i.e. the one closer to surface representation of a sentence, which is a text or utterance) express functions of units at the neighbouring higher layer (i.e. the one closer to the meaning of the sentence). The second feature is that FGD considers the syntactic relations between units in a sentence to be a set of dependencies, rather than a structure of constituents.

1.2.1 Description of the dependency structure

Dependency structure of a sentence is a rooted tree. Its nodes represent units of a certain layer and edges represent relation of **dependency** between two nodes. Dependency usually expresses what is called **(immediate) subordination**. The node closer to the root of a sentence is called **governor**, the other is called **dependent**. Further on, I will usually use alternative terms **parent** for governor and **child** for dependent, since not all the edges represent linguistic relation of subordination—some of them have rather technical character, e.g. edges containing nodes representing punctuation marks. By analogy, I will sometimes refer to node's grandparent, siblings, ancestors (transitive closure of the parent relation), offsprings (transitive closure of the child relation), and so on.

Besides, nodes in a dependency tree are linearly ordered according to a certain criterion. The criterion is specified in description of respective layers. In visualizations, parent–child relation is usually captured by the y-axis: a parent is higher than its child; and linear order of nodes is expressed by the x-axis: the left-to-right arrangement corresponds to ascending order.

An example of a sentence annotated using dependencies is shown in Figure 1.1.

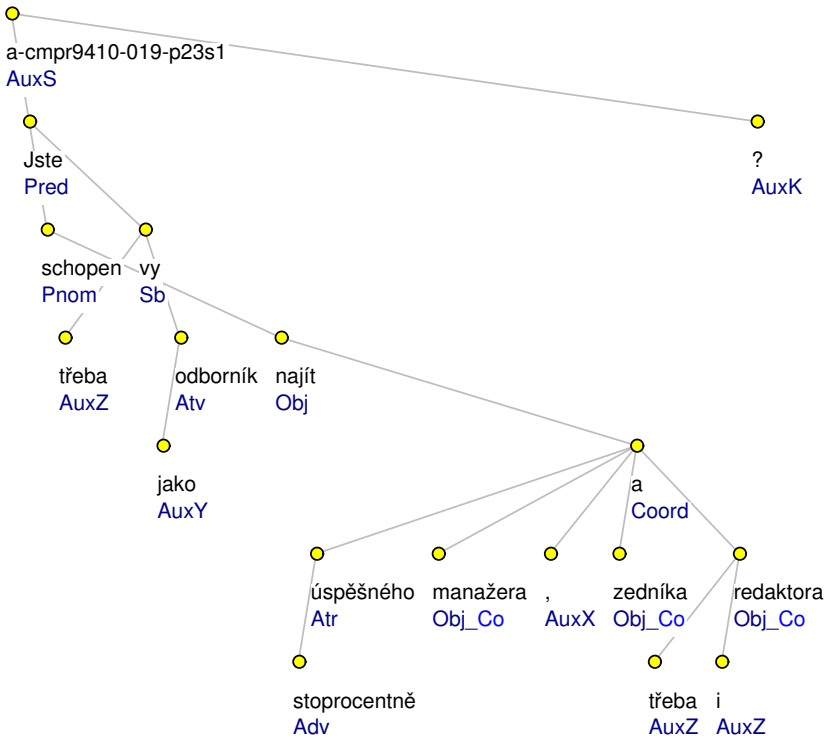


Figure 1.1: The a-layer annotation of the sentence *Jste schopen třeba vy jako odborník najít stoprocentně úspěšného manažera, zedníka a třeba i redaktora?* (*Are for instance you as a specialist able to find a hundred-percent successful manager, bricklayer, and even editor?*; lit.: *Are able for-instance you as a-specialist to-find hundred-percent successful manager, bricklayer and even also editor?*)

In a dependency tree, a language phenomenon of a non-dependency nature should be captured. **Coordination** is a relation among several tokens in a sentence where none of them is subordinated to any of the remaining ones—they rank equally. This "horizontal" relation, as opposed to the "vertical" relation of subordination, needs special treatment. The reason for this are technical difficulties which would occur if we captured coordinations by means of de-

pendency edges only: a common child of members of a coordination would then have more than one parent and the structure would not be a tree any longer.

The treatment of coordinations follows. An auxiliary token, usually a coordinating conjunction or comma, represents all members of a coordination, which are attached to it as its children. When each member of a coordination should be a child of a node, the node becomes the parent of the coordination node. Conversely, when each member of a coordination should be the parent of a token, the token becomes a child of the coordination node and treeness of a structure is retained. Members of a coordination are distinguished from their common children by their special attribute. In Figure 1.1, nodes *manažer*, *zedník*, and *redaktor* are members of a coordination (this is marked with their suffix *_Co*) and *úspěšný* is their common child.

I will refer to the nodes which would be the parents of a node if edges in the dependency structure corresponded to the relation of dependency only, as to its **effective parents**, whenever I will need to distinguish them from its direct parent. For example, in Figure 1.1 the effective parent of the node *manažer* is the node *najít*; the node *úspěšný* has three effective parents: *manažer*, *zedník*, and *redaktor*.

I use the term "coordination" in its broader sense and I also denote the relation of apposition with it. Linguistically, coordination and apposition differs in the manner in which they combine meaning, but, from the technical point of view, their topological structures are the same, so I address both these types of relations by one term.

There is also one phenomenon, which should be noted. It is a notion of **projectivity**. Several (equivalent) definitions of projectivity exist; one of them follows. An edge in a rooted tree having its nodes linearly ordered is **projective** if every node placed (according to the linear order) between the nodes forming the edge is their offspring; otherwise the edge is **non-projective**.[1] When a tree is depicted in the way described above, non-projective edges are exactly those being crossed by the perpendicular from at least one node. In Figure 1.1, only the edge *schopen—najít* is non-projective.

1.2.2 Prague Dependency Treebank

Prague Dependency Treebank, PDT (see [Hajič et al., 2006]), is a human annotated corpus of Czech texts obtained from the Czech National Corpus.[2] Sources of the text are mainly daily newspapers, the rest of the text comes

[1] Note that non-projectivities cannot be described by standard context-free grammars.

[2] `http://ucnk.ff.cuni.cz/`

from other press. The treebank is currently in version 2.0. It is annotated at three layers (see their description below). Not all the texts are annotated at all the layers; however, a principle is established that when a certain text is annotated at a certain layer, it is annotated at all lower layers as well.

The data annotated at a certain layer are always divided in the approximate ratio of 8:1:1 into training data, development test data, and evaluation test data. The amount of data annotated is given in descriptions of respective layers.

PDT 2.0 is being held in an XML-based format called PML, Prague Markup Language (see [Pajas and Štěpánek, 2005]). For my work, there are some important facts about it to be mentioned. Each layer of annotation of a file containing text is stored in a separate file[3], and a file corresponding to a layer contains links to the neighbouring lower layer (and other data resources, if any). Annotation of a node, when it is not expressed by its position in the structure of annotation, is captured in its attributes. Their description is given in the introduction to particular layers, which follows now.

1.2.3 Morphological layer

Plain text is first annotated at the lowest, **morphological layer** (**m-layer**). The text is segmented into sentences and **tokens**, which can be a word, a number expressed by digits, or a punctuation mark. To each token (stored in the `form` attribute), its lexical entry and values of its morphological categories (part of speech, person, number, tense, gender, voice, ...) are assigned. The lexical entry, called **lemma** (and stored in the `lemma` attribute) is, for the sake of readability, usually expressed as a basic form of the token, e.g. in the nominative of singular for nouns. Values of morphological categories are merged into one string (stored in the `tag` attribute) and I will refer to it as to **m-tag** further on. Guidelines for annotation at the m-layer are given by [Zeman et al., 2005].

Since there are as many as 13 morphological categories[4], the number of unique m-tags appearing in the treebank is over one thousand (of about 4,000 possible). I describe elsewhere how I dealt with this sparseness of data. However, I sometimes compare my results with a certain approach described in [Collins et al., 1999], or I even use it. The authors, having dealt with the described problem for the sake of training a parser, proposed a very good compromise (although in rather *ad hoc* manner) between the sparseness of m-tags and their descriptive power. In general, they reduced an m-tag to just two characters: the first of them being the part of speech, the second one being a case if applied (with nouns, adjectives, pronouns, numerals expressed by words, and

[3] This is usually called *stand-off annotation*.

[4] Of course, for particular parts of speech only some of them are relevant, e.g. for nouns values of at most 7 categories are filled.

prepositions), or a detailed part of speech otherwise. I refer to this approach as a **two-letter m-tag**.

1.2.4 Analytical layer

At the **analytical layer** (**a-layer**), a sentence is represented as a dependency tree described above. Each token is represented by exactly one node and no node is added except for an extra root node, which has technical character and which the predicate of a sentence and the final punctuation depends on. Left-to-right order of tokens in the tree corresponds to their order in the sentence (and is explicitly given in the `ord` attribute).

In addition, several attributes are assigned to each node at this layer. The most important one is its **analytical function** (stored in the `afun` attribute), capturing the type of the dependency relation between the node and its parent.[5] Since this is a label denoting the syntactic function of the token, I refer to it as to **s-tag** for short. There are 23 possible s-tags plus 3 special ones for cases when the sentence is syntactically ambiguous in a special way the details of which are beyond the scope of this thesis. A node also carries information whether it is a member of a coordination (stored in the `is_member` attribute).

The total amount of data annotated at the a-layer is 1,504,847 tokens in 87,980 sentences.[6]

An example of a sentence annotated at the a-layer is in Figure 1.1. Guidelines for annotation at the a-layer are given by [Hajičová et al., 1999].

1.2.5 Tectogrammatical layer

The highest layer is the **tectogrammatical layer** (**t-layer**). Guidelines for annotation at this layer are given by [Mikulová et al., 2005]. The t-layer captures the deep (underlying) structure of a sentence and its purpose is to describe the linguistic meaning of the sentence. Similarly to the a-layer, the structure is captured in the form of dependency trees. It has the following characteristics:

- Nodes represent only autosemantic words (and, for technical reasons described above, heads of coordinations).

[5] Strictly speaking, it should be the label of the corresponding *edge* instead of the node.

[6] It should be noted that all experiments with PDT 2.0 were performed on its prerelease version. The final version, which originated after the experiments had been done, differs only slightly, though: it contains 19 files less, which were duplicated (out of 7,129 files).

- Synsemantic (auxiliary) words and punctuation marks are not represented by nodes, they may only affect values of attributes of autosemantic words to which they are related to (e.g. prepositions to nouns).
- If a unit is present in the meaning of a sentence but has not been expressed in it (on its surface), a node representing this unit is added. Nodes are inserted mainly in the following two cases: when filling in an ellipsis of words already presented at another place of the sentence, and when valency dictates so.
- Edges represent relations between units of meaning.
- The left-to-right order of nodes follows the information structure of the sentence (Topic-Focus Articulation).

The annotation at the t-layer is very complex—there are as much as 39 attributes at the nodes. I will describe the most important attributes or groups of attributes briefly. Some of them have rather technical character: but I will go into certain details even in their description, since they are important for my work.

- **Functor** (stored in the `functor` attribute), similarly to the s-tag at the a-layer, captures the tectogrammatical relation of a node relative to its governor.
- Type of nodes is stored in the `nodetype` attribute. It has rather technical character, it defines which attributes the node has and how its children are interpreted.
- **Grammatemes** is the whole set of semantically oriented attributes more or less corresponding to several morphological categories. A grammateme is filled in cases when the appropriate morphological category has its value not for syntactic reason (e.g. to express agreement), but when its value is relevant for the meaning of the sentence. Number and degree of comparison are examples of grammatemes having their morphological counterparts; deontic modality and type of indefiniteness are examples of the other ones.
- The semantic part of speech is stored in the `gram/sempos` attribute and its value thus defines which grammatemes are to be filled for the given node.
- (Deep) order of a node (`deepord`) determines its left-to-right order in its tectogrammatical tree.
- The `t_lemma` attribute means "tectogrammatical lemma" and is usually equal to the lemma (from the m-layer) of the corresponding token; however, sometimes it differs, mainly due to the fact that some semantic

information from lemma can be extracted into grammatemes and tectogrammatical lemma thus can have even more basic form. For example, pronoun *všichni* (*all*) can be annotated as *kdo* (*who*) having its grammateme of indefiniteness set to value meaning "total". When a node does not originate from the a-layer, the attribute has a special value indicating that it represents a general participant, or that the node is the subject of coreference, or that it represents an "empty" governing verb predicate etc.

- For connecting the t-layer with lower layers, links to corresponding nodes at the a-layer are stored in three attributes: a link to the corresponding autosemantic word, if any (`a/lex.rf`)—I will refer to it as to **a-link**; links to corresponding synsemantic words[7], if any (`a/aux.rf`); and finally a link to the root of the corresponding tree (`atree.rf`; defined only for roots).

- In the `val_frame.rf` attribute, the identifier of the corresponding valency frame, which is kept in a separate valency lexicon, is stored.

- The `tfa` attribute bears information about the so called information structure of a sentence—it is recorded there whether the node is a part of the topic, the contrastive topic, or the focus of the sentence.

- The `is_member` attribute determines whether the node is a member of a coordination.

- The `is_generated` attribute expresses whether the node is new at the t-layer. When set, it does not necessarily mean that the node has no counterpart at the a-layer, since it can be a "copy" of another t-layer node and more nodes can refer to one a-layer node through their `a/lex.rf` attribute.

- In the `coref_gram.rf`, `coref_text.rf` and `coref_special` attributes grammatical and textual coreference is captured.

- The `id` technical attribute carries the unique identification of a t-node.

The total amount of data annotated at the t-layer is 833,357 tokens in 49,442 sentences.

The sentence in Figure 1.1 is now annotated at the t-layer in Figure 1.2.

[7] For some reason, links to punctuation are not preserved.

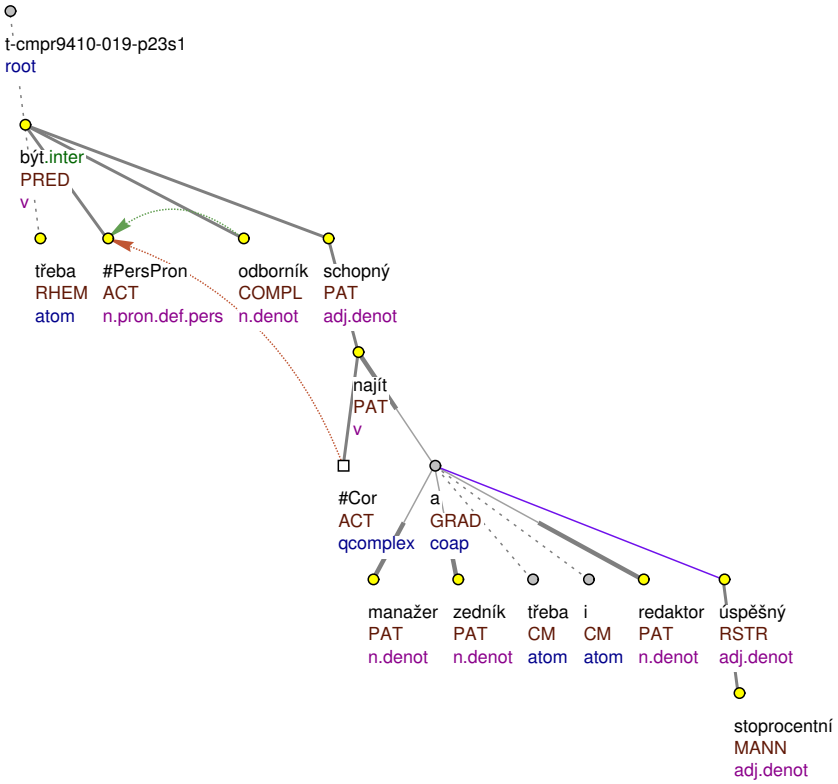


Figure 1.2: The t-layer annotation of the sentence from Figure 1.1

1.3 Transformation-based learning

1.3.1 The basis of the method

In almost all the tasks in this thesis I use the technique named **transformation-based learning**, **TBL**. It was introduced in [Brill, 1992] as one of the methods of rule-based machine learning and since then it has been successfully used for various tasks in the area of natural language processing, e.g. part-of-speech tagging ([Brill, 1992]), constituency parsing ([Brill, 1993]), dependency parsing ([Ribarov, 2004]), prepositional phrase attachment, spelling corrections, and phrase chunking. The comprehensive list can be found in [Ngai and Florian, 2001].

The main idea of transformation-based learning is to learn an ordered list of rules improving the state of annotation of the training data. The rules are

being learnt greedily and the learning stops when no further improvement can be reached.

At the beginning of training, we initialize one copy of the training data with a simple annotation (e.g. for part-of-speech tagging, it can be the most probable tag for the given word) and these data become the starting training data. In each iteration of training, all possible rules from a prespecified set are considered and the rule which improves the annotation of the current training data most significantly, compared to their golden annotation, is added to the list of learned rules. This rule is then applied to the training data, which results in their annotation being more accurate.

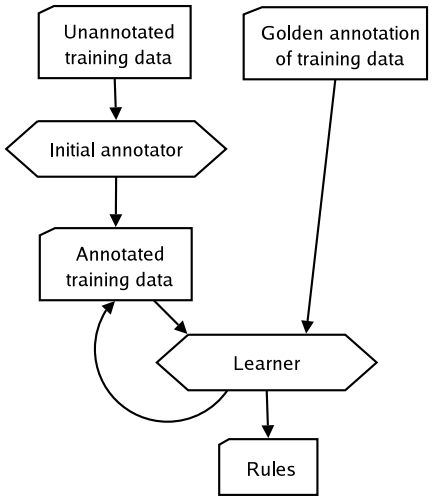


Figure 1.3: The flow of data in the learning phase of TBL

The most improving rule for the annotation is the one maximizing the decrease in number of errors, i.e. maximizing the function defined as the number of errors corrected by the rule (good applications) minus the number of errors newly made by the rule (bad applications). Unlike in many other algorithms, this directly optimizes the evaluation function (given as a number of correctly annotated samples) rather than some hopefully correlated metrics like entropy.

The process of annotation of new unseen data is then simple: the learned rules are applied to them in the order they were learnt.

The transformation-based learning is claimed (by [Ngai and Florian, 2001]) to have, e.g. when compared to methods based on probabilistic models, several advantages:

- it is a very flexible method,

- linguistic features are captured in a small and concise set of rules,
- it is stable, and
- it cannot be overtrained easily.

However, a direct implementation of this algorithm results in lengthy learning time. Some techniques for accelerating the learning have been proposed (e.g. [Satta and Brill, 1996]) and several of them have been implemented (see their comprehensive list and comparison in [Ngai and Florian, 2001]). One of the successful implementations is the *fnTBL* toolkit—it was used for solving the tasks in this thesis and it is described below.

1.3.2 The *fnTBL* toolkit

The ***fnTBL***[8] toolkit was introduced in [Ngai and Florian, 2001]—see this source for details of almost all the information given in this section. It is, to my best knowledge, the fastest implementation of the TBL concept: on large corpora it performs about 2 orders of magnitude faster than a standard implementation and, besides, it does not consume as much memory as some of its competitors.[9]

The idea of the speedup brought by *fnTBL* is that counts of good and bad number of applications of rules are not regenerated in every step; instead, they are stored and updated when the best rule is applied. The advantage is that only counts of rules which would apply in neighbourhood of data altered by the best rule need to be updated. Besides, these rules are obtained by generating them from the data according to templates instead of seeking through all the rules.

While TBL is a general concept, *fnTBL* is aimed at classification tasks only. This brought certain difficulties, since some of the tasks I had to solve are not classification tasks nor can be directly transformed into them. How transformations of my tasks into the form workable by *fnTBL* look like is stated at the respective places.

The toolkit can solve tasks where samples are both independent (e.g. PP-attachment) and interdependent (e.g. POS tagging). Each sample of data consists of several features. The rules which the toolkit tries to learn are specified by **rule templates** which have to be designed before the learning. A rule template is a subset of the names of possible features together with the name of a feature which carries the identification of the class the sample belongs to

[8] `http://nlp.cs.jhu.edu/~rflorian/fntbl/`

[9] We will see later that even so I had to reduce memory requirements somehow.

(since it is possible to perform more classification tasks at once, see also [Florian and Ngai, 2001b]). A rule is an instance of a template: particular values are assigned to all the features. The rule is interpreted in the following way: a sample belongs to the given class (for the given classification task) if the given features have the given values—this expresses the conjunction of predicates concerning values of features. A rule can refer to values of features of the current sample or, in the case of interdependent samples, to those of the neighbouring ones—the neighbours are given as a fixed integer expressing the distance from the current sample. It is also possible to check for the presence of a value of a particular feature in a sequence of samples, or to check for a value in a set of features of a sample—this expresses a limited set of disjunction of predicates concerning values of features. An example of a rule from POS-tagging task together with its corresponding template follows.

```
POS_-1=DT POS_0=VB => POS=NN
POS_-1 POS_0 => POS
```

The toolkit is capable of determining the probability of the reliability of classifications (described in [Florian et al., 2000]); however, the authors state that the implementation is buggy.

The toolkit has many options and settings. Some of them affect generating of rules; the most important ones of those I experimented with follow.

- `threshold`: the algorithm stops when the best rule has the given score (i.e. number of good applications minus the number of bad applications); default value is 2;
- `ORDER_BASED_ON_SIZE`: determines which rule is chosen when more rules with the same score meet: the one consisting of more atomic predicates (default), or the one consisting of less atomic predicates, or the one whose template is declared first;
- `allPositiveRules`: if set, outputs all the rules having no bad application and the number of good applications greater than the given number at the end of training.

The toolkit consists not only of the learner and classifier, but other utilities are provided as well, e.g. those creating initial annotation by creating and applying a lexicon of the most probable classes for a combination of features.[10]

More information about (not only) technical matters concerning the toolkit can be found in its documentation, [Florian and Ngai, 2001a].

[10]The authors recommend always to do the initial annotation, because it reduces time and memory requirements. For my tasks the reduction was 50–70% of time and about 40% of computing memory.

1.4 The claim of language independence

As stated above, my tool should be language independent. However, the annotation manuals describe the annotation of Czech texts and it is not clear how the annotation principles would differ especially for languages that are typologically distant from Czech—it is only clear that even the most language-independent layer, the t-layer, would differ.

Moreover, experiments with another languages could not be performed, since besides PDT, no corpus human-annotated at the t-layer which would be big enough to be used for training has existed up to now.[11]

In this situation I can do only two things: not to hand-code specifics of Czech and to list which attributes of the processed data the tool relies on, be they those used during training (i.e. features of training samples), or those assigned by training, or those assigned by hand-coded procedures.[12] These lists become important if the tool is to be adjusted for another language. For the analytical analysis, these lists are hopefully clear from the text of the respective chapter and thus they are not given explicitly; for the tectogrammatical analysis, these lists are given in Section 3.1.

1.5 The scope and division of the work

According to Section 1.2, we can view the task the tool has to perform as a sequence of morphological, analytical, and tectogrammatical analysis (regardless of the way the tool will be implemented in: whether such sequence is performed, or whether the whole analysis is carried out at once).

For the reasons listed below, I decided not to include the morphological analysis into my tool; thus, its input is a morphologically annotated text instead of a plain text.

- The m-layer is most language dependent from all three layers and we can hardly hope to find a universal set of features of word forms capturing their morphological characteristics well, when we consider that even just in the mentioned languages, there can be expressed by as various

[11] There is only sample of about 500 English sentences annotated at layer similar to the Czech t-layer (see [Kučerová and Žabokrtský, 2002]), which is part of the Prague Czech-English Dependency Treebank 1.0 (`http://ufal.mff.cuni.cz/pcedt/`).

[12] Intuitively, some categories can be considered as more language independent (e.g. `is_member`) than other ones (e.g. `lemma`), so I take the liberty to hand-code the assignment of some of the former ones instead of training them.

means as word affixes, prefixes, and changes of root vowels, disregarding all possible irregularities.

- At least for highly inflectional and agglutinative languages, there is a better way to annotate a corpus at the m-layer than to manually fill m-tags and lemmas of each token in an annotated text. A lexicon is created where for each token all possible pairs of lemma and m-tag that may have generated the token are given, regardless the context of the token; and then only the correct pair is chosen, where more than one is proposed.[13] In other words: in order to obtain an annotated corpus for training purposes, it is useful to develop a morphological analyzer based on hand-written rules—but then, obtaining another morphological analyzer (by means of training) is obviously of little use.

I also decided to implement the analytical and tectogrammatical analysis in separate steps; the reasons for doing so are summarized below.

- The primary goal was to ease my work by splitting it into two smaller parts. The a-layer seems to be a good intermediary between the m-layer and the t-layer: it only brings the dependency structure into m-layer; on the other hand, this structure is already very similar to that of the t-layer.
- The by-product of the splitting is a parser, a tool assigning the structure at the a-layer[14], and the parser itself can be used for several tasks (see their list at the beginning of Chapter 2).
- The splitting of the work and the independence of its parts brings higher flexibility as well: its most important consequence is that one can use e.g. another (analytical) parser together with my tectogrammatical analyzer.[15]

The structure of the thesis reflects the independence of the analytical and tectogrammatical analysis: Chapter 2 deals with the first one, Chapter 3 with the latter one, and each of the chapters contains its own description of evaluation, comparison with previous works etc.—and, of course, overview of its sections. Chapter 4 concludes results described in both previous chapters.

My tools are downloadable from the following web address: `http://ufal.mff.cuni.cz/~klimes/phd/`

[13] This is also exactly the way how PDT was annotated at the m-layer. The process of selection of the correct pair is called *morphological disambiguation*.

[14] This statement is not exact; for a difference between parsing and the analytical analysis see the beginning of Chapter 2.

[15] This is, due to lower performance of my analytical parsers, highly advisable.

Chapter 2

Analytical analysis

'Would you tell me, please, which way I ought to go from here?'
'That depends a good deal on where you want to get to,' said the Cat.
'I don't much care where—' said Alice.
'Then it doesn't matter which way you go,' said the Cat.
'—so long as I get somewhere*,' Alice added as an explanation.*
'Oh, you're sure *to do that,' said the Cat, 'if you only walk long enough.'*
[Carroll, 2003]

Analytical analysis, i.e. analysis which has the annotation at the a-layer (described in Section 1.2.4) as its output, can be divided into two main parts. One of them is **syntactic analysis**, usually called **parsing**, which is a procedure of building a syntactic tree structure out of a string of words and other symbols, and thus specifying relations among these words. There are several reasons to parse text. One of the obvious ones is that a parser can be the core of a grammar checker—the tool checking whether the occurrence of a word in a certain place of a sentence is plausible. A parser is useful within the task of speech recognition as well, since it can correct some bad decisions of the acoustic model employed in this task. Last but not least, a parser can be employed by annotators assigning syntactic structures (which can be then used e.g. for training tools performing the mentioned tasks) to preannotate the data and thus to ease their work. A comprehensive description of what a parser is good for can be found in [Zeman, 2005].

The other part of the analytical analysis is the assignment of an s-tag to each token, which describes the syntactic function the token has in the sentence.

This chapter describes two parsers I developed and the procedure assigning the s-tags. Both parsers need an annotated corpus for creating their model, which they then use for parsing. Although their target language is primarily

Czech, at least the latter is intended to be independent of the language being processed.

Section 2.1 describes the evaluation method used for reporting the parsing results. There is a review of parsers of the Czech language in Section 2.2. Sections 2.3 and 2.4 describe both methods the parsers are based on and Section 2.5 compares them. Section 2.6 introduces the procedure for assigning s-tags.

2.1 Evaluation

All evaluation in this work is done using test data, which are distinct from training data. One of their copies is annotated manually—we take it as the measure of correctness and we call it the **golden annotation**. The other copy being the subject of a comparison is called the **test annotation**.

For an evaluation of my task, which is analytical dependency parsing, I use the most common metrics—(dependency) **accuracy**. It is defined as the number of correct dependencies, i.e. those assigning a node to the same parent in the test annotation as in the golden annotation, divided by total number of dependencies in one of the annotations.[1] Below, we will denote accuracy as A.

For Method 1, when a parser may be unable to assign the parent of a node at all, we may be interested in other characteristics of its performance as well. We define **coverage**, C, as the ratio of the number of assigned dependencies to all dependencies, and **precision**, P, as the ratio of the number of correctly assigned dependencies to all assigned dependencies. Accuracy is defined in the same way and it is the product of coverage and precision in this case.

Since I was performing experiments with analytical parsing when PDT 2.0 was not available, the results are reported on PDT 1.0 ([Hajič et al., 2001]). Its data consist of 1,507,333 tokens in 98,263 sentences and they are divided into training set, development test (d-test) set, and evaluation test (e-test) set with the approximate ratio of 10:1:1. All tools whose results I report are trained on all the training data of PDT 1.0 annotated at the a-layer and they are tested on all its d-test data annotated at the a-layer. The m-layer of the whole data is disambiguated automatically.

As late as all the experiments with analytical analysis were done, PDT 2.0 has became available and I wanted to test the performance of my tools on it. The experiments were not redone for PDT 2.0, however; I only ran the

[1] From the description of the a-layer it follows that all annotations of the same data have to contain the same nodes and thus the same number of dependencies, which is equal to the number of nodes.

parsing Method 2 and the process of assigning the s-tags on PDT 2.0 with the settings which proved to be the best for PDT 1.0. The results are reported at the respective places together with those obtained using PDT 1.0. As in the case of PDT 1.0, all mentioned tools are trained on all the training data of PDT 2.0 annotated at the a-layer and tested, unless stated otherwise, on all its d-test data annotated at the a-layer; the m-layer data used for these tasks were disambiguated automatically.

2.2 Previous works

There are several parsers of the Czech language. In order to be able to compare my results with their ones, I mention only those for which results on d-test data of PDT 1.0 are available.

These parsers fall into two groups: one of them contains those developed originally for English with its constituency-based structures, which are then converted into dependency structures; the other group contains parsers working with dependencies internally. The existing parsers are described in Table 2.1 and their performance on PDT 1.0 (and, for some of them, on PDT 2.0) is given as well. Some of these results are obtained from web page `http://ufal.mff.cuni.cz/czech-parsing/`

However, the most successful parsers of the Czech language originated as the combination of some of the described ones: on PDT 1.0, the one described in [Zeman and Žabokrtský, 2005] achieved the accuracy of 86.3%; the combination described in [Holan and Žabokrtský, 2006] yields the accuracy of 86.2% on PDT 2.0.

2.3 Method 1: Optimal context length

This section describes my early experiments with analytical parsing and attempts to develop a method of machine learning based on human-readable rules. Although the method is usable only in parsing and possibly in some other areas where linear context of a token can help significantly, and even in these areas it exhibits some deficiencies, it brings an original approach to machine learning using rules, which builds its model on sequences of symbols. Besides, some ideas are used in a more successful transformation-based method described in Section 2.4.

Author	Short description and references	PDT 1.0	PDT 2.0
Michael Collins	originally a constituency-based parser; based on lexicalized probabilistic context-free grammar ([Collins et al., 1999])	82.6%	82.4%
Eugene Charniak	originally a constituency-based parser; inspired by maximum-entropy ([Charniak, 2000])	84.3%	
Kiril Ribarov	dependency parser; perceptron-based [Ribarov, 2004]	about 72%	
Daniel Zeman	dependency parser; statistical modeling dependencies as word bigrams ([Zeman, 2005])	74.7%	75.0%
Tomáš Holan	dependency parser; based on a push-down automaton and using genetic algorithm to learn weights of rules ([Holan, 2005])	73.9%	74.0%
Ryan McDonald	dependency parser; searching for Maximum Spanning Tree ([McDonald et al., 2005])	84.4%	84.2%
Keith Hall and Václav Novák	improvement of Charniak's parser by means of recovering non-projectivities ([Hall and Novák, 2005])	85.0%	
Zdeněk Žabokrtský	dependency parser; hand-written rules tailored for Czech ([Holan and Žabokrtský, 2006])	75.2%	76.1%

Table 2.1: Description and performance of existing parsers

2.3.1 The basis of the method

The language model of this parsing method has the shape of rules describing dependencies among tokens in a sentence. The method is based on the idea that in order to be able to assign the parent of a token, we (usually) do not need to know the whole sentence but it is sufficient to know only the important information from the context of the token in question. Thus, the first problems to be solved are which part of the information is important and what does "context" mean.

When considering the context of a token, I work with the linear surface structure of the sentence only, neglecting structure of the rising dependency tree for simplicity. This approach cannot obviously profit from the knowledge of dependencies of the neighbouring tokens, although, on the other side, it cannot get confused when the dependencies are not assigned correctly. I define the

context of a token as a continuous sequence of tokens from the linear surface structure which include the token in question. For linguistic reasons, I consider only the contexts in which the parent of the token is also included: when the possible parent is out of the context, we have almost no information about it, which is too little to decide whether this token is the parent. The algorithm itself determines the optimal length of the context.

Since a context can contain an unlimited number of tokens, little information about tokens can be processed only so that the computation can fit into memory. For this reason, I use a two-letter m-tag as the only information about a token.

The issue of identification of the parent of a token contained in a rule has rather technical character and I encode the parent as its relative distance from its child token. As a special case, when the parent of a token is the root node of the sentence, zero is encoded as the distance and interpreted in the described way.

A rule thus contains the tag of the token in question, the tags of the tokens in its context in the order they appear in, and the distance of the parent. Technically, the first and the second entry are delimited with a colon and the second and third one with the "greater-than" sign. In the notation of the context, several special symbols are defined as well: `'^^'` means the beginning of a sentence, `'$$'` denotes its end, and `'__'` indicates the position of the token in question. For example, the rule `N2: N4 A2 __ > -2` has the following meaning: when a noun in accusative followed by an adjective in genitive and by a noun in genitive occurs, the parent of the noun in genitive is the noun in accusative.

2.3.2 Inference of rules

Having decided about all other substantial aspects of the method, we can decide now how rules will be inferred. My idea was to develop a system where once a rule is applied to a configuration, no other rule will be applied to the same configuration, i.e. one rule only assigns the parent to a token. For this reason, the parsing procedure is simple:

- for every input sentence
  - for each of its tokens
    - go through the rules in the order they have been learnt in and apply the first applicable rule

A rule is applicable when the tag of a token in question, together with the tags of tokens in its context, as recorded in the rule, matches the sample.

From the sentence symbolically written as `Db Vp N1` (the sequence of an adverb, a verb in present tense, and a noun in nominative) where the verb is the parent of both remaining nodes, we can infer the following rules in an obvious manner.

```
Db: ^^ __ Vp N1 $$ > 1
Vp: ^^ Db __ N1 $$ > 0
N1: ^^ Db Vp __ $$ > -1
```

However, these rules are very specific and thus of little use, and there is a need to find more general ones. For this reason I regard the rules inferred in this way as my training samples and based on them I find rules which are as general as possible (i.e. which have the shortest possible context), but which unambiguously determine distance to the parent when applied to all training samples. The rules are generated starting from those with shorter context—with the condition that the context recorded in a rule contains the parent being assigned by the rule—and when a rule cannot assign a parent unambiguously, the rule is crumbled into several more specific rules by extending the context. For simplicity, I consider only rules with continuous contexts.

The training procedure, based on the ideas above, follows.

- infer training samples from the training data
- delete those samples where the distance is ambiguous[2]
- NEXT_TAG: for every child tag
  - NEXT_LENGTH: for increasing context length starting from 1
    - infer the rules whose contexts have the current length and contain the parent from training samples
    - determine how many times each of such rules would apply correctly and incorrectly in the training samples
    - choose and record the best rule, i.e. the one applied correctly the most times and never incorrectly
    - when such a rule does not exist, go to NEXT_LENGTH[3]
    - delete samples which the best rule was applied to[4]
    - if no more training samples exist, go to NEXT_TAG

There is one matter remaining to be solved yet. The analytical structure is a tree; however, since the parents of tokens are assigned independently, the output structure may contain cycles. I chose to solve this problem by removing

[2] For obvious reasons, when there are two training samples differing only in the parents, no rule assigning the parent correctly can be generated from them.

[3] This is how the rule is crumbled.

[4] As during parsing, a sample is not considered any more once the parent has been assigned.

one edge between a child and its parent in every cycle. The best criterion for the edge to be removed seems to choose the one corresponding to the rule which was applied the least times (to the training data) and which can be thus considered to be the least reliable. For this reason, there is a need for storing the number of applications of a rule together with the rule.

With the method described I achieved $C = 80.2\%$, $P = 77.4\%$ and $A = 62.1\%$, having 600,981 rules (there are 1,224,076 dependencies in the training data).

2.3.3 Inference with exceptions

However, the system of rule inference is not optimal, since because of exceptions (and errors in the annotation), a rule crumbles into a number of less general rules. This can lead to a lower coverage (and even precision, since the rules are less linguistically motivated). The phenomenon can be illustrated on the following example: because of the `A1: A1 J^ __ N1 > -1` rule (`A1` means an adjective in nominative, `N1` is a noun in nominative, `J^` is a coordinating conjunction), the `A1: __ N1 > 1` rule is crumbled into rules `A1: Vc __ N1 > 1`, `A1: P6 __ N1 > 1` etc., while many other quality rules, e.g. `A1: P3 __ N1 > 1`, are not inferred at all, because the corresponding configurations were not seen in the training data. To solve this problem, it is sufficient to modify the training algorithm so that it finds exceptions to the rule and writes them before the given rule—in our example the correct order should be `A1: A1 J^ __ N1 > -1` and `A1: __ N1 > 1`. The parsing algorithm already has the desired property to stop when the first rule is applied, and thus it can remain unchanged.

For our algorithm, we have to formalize what an exception is. An exception to a rule is a rule which applies correctly or does not apply at all when the original rule would apply incorrectly, and which does not apply when the original rule would apply correctly.

The training algorithm given below considers the formalization; it also allows for exceptions to an exception etc.

There is one more issue to be solved: to decide when to consider a rule to be correct, although with exceptions, and when to consider it to be incorrect. I defined this criterion as a certain ratio of correct applications of a rule to incorrect applications—when the actual ratio is greater than or equal to this threshold, exceptions to the rule are generated, otherwise the rule possibly crumbles as described above. Experiments show that the best ratio is 2:1.

The new training procedure is described below.

- infer training samples from the training data

- delete samples where the distance is ambiguous
- NEXT_TAG: for every child tag
  - NEXT_LENGTH: for increasing length starting from 1
    - infer the rules whose contexts have the current length and contain the parent from training samples
    - delete rules not meeting the above requirements (this step is not applied for the first time)
    - determine how many times each of such rules would apply correctly and incorrectly in the training samples
    - choose the best rule, i.e. the one applied correctly the most times and with the correct-to-incorrect ratio above the threshold
    - when such a rule does not exist, go to NEXT_LENGTH
    - if the best rule was applied incorrectly at least once
      - mark training samples where the best rule was applied correctly and where it was applied incorrectly
      - call this functionality (except for the inference of training samples) recursively, but on the marked samples only[5]
    - record the best rule
    - delete samples which the best rule was applied to
    - if no more training samples exist, go to NEXT_TAG

For the algorithm with exceptions implemented, I achieved $C = 87.7\%$, $P = 79.5\%$ and $A = 69.7\%$ ($\Delta A = 7.6\%$), having 355,608 rules.

We can also state one more confinement to exceptions: an exception does not apply when the original rule does not apply; however, this stricter requirement leads to greater number of rules and thus, for the reasons described above, to slightly worse results.

2.3.4 Connecting the dangling nodes

When striving for the best accuracy, we can try to find parents which our algorithm was not able to assign. For this reason, I developed a simple back-off procedure which operates in the following manner. I separated a small part of the training data, which now serves as held-out data; and parsing is trained on the remaining training data only. The held-out data are parsed with the rules obtained and statistics are gathered from the tokens having no parent assigned: the probability of such situations are computed for these tokens in which the token has another token as its parent when this possible parent occurs in the sentence. Child and parent tokens are characterized by their two-letter m-tags; parent tokens are characterized by the information whether they occur to the left or to the right of the child token as well.

[5] This serves for inference of exceptions to the best rule.

After the parsing, the analysis of tokens with non-assigned parents runs as follows: from possible parents occurring in the sentence the one is chosen which has the highest probability recorded. From all the tokens with the same characteristics, only the one that is the closest one to its possible child can be chosen. In practice, it also turns out that adding a penalty for a big distance between a child and its parent improves the results—for each position between them the recorded probability is reduced by 1/15.

Results produced by this procedure are less reliable than those of the parsing procedure (see drop of precision below); therefore, when breaking cycles, lower weights are artificially assigned to edges established by this procedure.

After adding this simple procedure I achieved $C = 97.9\%$, $P = 74.9\%$ and $A = 73.3\%$ ($\Delta A = 3.6\%$).[6]

2.3.5 Deletion of unreliable rules

It turns out that the rules which were applied to the training data only once are too unreliable: the procedure described above can assign parents better than these rules. When these rules are deleted from the rule file and only the remaining rules are applied, the results are slightly better: $C = 98.4\%$, $P = 74.7\%$, $A = 73.6\%$ ($\Delta A = 0.3\%$). These are the final numbers of this method. Besides, the number of rules significantly decreases to 65,659.

2.3.6 Possible improvements

If somebody wants to improve the accuracy, the possible way may be to increase the generalization of rules somehow and thus, hopefully, to raise the accuracy of the method. There are at least two approaches possible. One of them is not to define context as a sequence of tags but rather as an expression operating on such a sequence; for example, regular expressions seem to be very suitable for this task ([Zeman, 2001]). The other approach makes the rules express an agreement between a child and its parent (and, possibly, other tokens in the context).

[6]The coverage is lower than 100% since the back-off procedure can assign a parent to a token only when the token occured in the held-out data, it had no parrent assigned, and one of the tokens occuring in the current sentence should have been its parent.

[6]For the sake of completeness: when in this situation the back-off procedure is left out, the accuracy drops greatly: $C = 82.9\%$, $P = 82.5\%$, and $A = 68.4\%$ ($\Delta A = -5.2\%$).

2.4 Method 2: Transformation-based classification

This parsing method is based on similar ideas as Method 1; however, I did not implement my own algorithm inferring rules that describe dependencies, but I left this task to *fnTBL*, one of the toolkits performing transformation-based learning (see Section 1.3.2). The result of this is a high chance of affecting the choice of information important for parsing, lowering amount of programming work and improving accuracy of the parser.

2.4.1 The basis of the method

Similarly to Method 1, the basic idea of parsing is to assign the parent to each token independently of other dependency edges and to use only the information from the context of the token for doing so. Unlike Method 1, we can easily design shapes of contexts (although it is not possible to have contexts of unlimited length any more) and choose what information from a context will be used.

Since *fnTBL* is capable of solving classification tasks only, there is a question how to transform the task of determination of the parent to a classification task. The answer can be that we can classify a token into a class identifying its parent. The identification should be unique, i.e. it should choose (at most) one token as a parent, but it should, because of reliability, also describe a possible parent by means of taking some of its properties into account—the more linguistically motivated the description is, the higher reliability of resulting rules can be expected. However, the description of a possible parent cannot be *too* specific, since otherwise it could be impossible for a rule to ever find a token with the given properties. A trade-off solution is needed.

It proved good to use two-letter m-tags for the task of description. To make the description unique, I append to its beginning the distance of the parent from its child, considering only tokens having the same tag as the parent has. E.g. `-1N4` means "the previous noun in accusative" and `+2VB` means "the second verb in present or future tense to the right", with value `0` meaning the (technical) root of a sentence.[7] This way of identification also solves the problem that, unlike Method 1, we cannot force the context in a rule to contain the parent, and thus we know almost nothing about it when it is beyond the context.

[7] Only in about 3% of cases, the parent is the second or even more distant token with the given tag.

2.4.2 Connecting the dangling nodes and enforcing trees

The fact that the parent assigned by a rule may be absent from the context of the rule also implies that a rule can assign a non-existing parent. For example, a rule can assign `-1VB` as the parent of the current token; however, there is no verb to the left of it. For this reason I developed a procedure which reads the parsed file and the rule file and places the dangling nodes: the effects of rules assigning a non-existing parent to a token are undone, so that the node gets the parent assigned by the latest rule from those assigning existing parents.[8]

As with the previous method, the resulting structure not necessarily is a tree. This problem is solved by removing one of the edges forming a cycle as well: the edge established by the latest rule is chosen and the effect of this rule is undone (i.e. the child node of this edge will have the parent appointed by the last but one rule which applied to this child). It also helps to consider edges containing children which were originally dangling to be less reliable, and thus prefer these edges for removing when they occur in a cycle.[9]

The described criterion for choosing an edge to be removed turns out to be the best. I also trained the probabilities of dependencies between tokens (similarly to the method described in Section 2.3.4) on held-out data and used them when deciding which edge to remove, but there has been no significant difference in the results; I got similar results when I removed the longest edge (measured by the linear distance of the two nodes forming it). Other approaches, like choosing the edge selected by the oldest rule instead of the latest one or choosing edges randomly, were found hurting the results a bit (the difference between the best and the worst from all the given approaches is $\Delta A = 0.3\%$).

2.4.3 The design of rules

There are many ways of creating the initial analytical structure—I experimented with so called "umbrella shape" (each node hangs on the root of its sentence) and "left string" (each node hangs on the previous node and the first node hangs on the root); these and several others can be found in [Zeman, 2005]. However, when using *fnTBL*, the best way is to initialize its input data in such a way that the initial assignment of classes is as correct as possible: one cannot expect higher accuracy, but the training is faster and it uses less memory. This is why I set the initial class of every token (deciding according

[8] Tests show that about 4% of nodes were dangling and more than one tenth of them has been placed correctly by the procedure.

[9] Probably the best solution of this problem would be searching for the Maximum Spanning Tree, as described e.g. in [McDonald et al., 2005].

to its two-letter m-tag) to the which it occurs most frequently in the training data in. This way I reduced the memory requirements and thus it was possible to add other templates into the template file.[10] This improved the accuracy of the parser as compared to the simple umbrella-shape initialization.

When designing the rule templates, I did not succeed in incorporating a lemma, which is the other type of information obtained from the m-layer, into templates containing m-tags. Rules containing lemmas of two tokens did not fit into memory, and sets of rules with a lemma of one token turned out to be less accurate than those containing only m-tags: after removing lemmas, memory requirements have decreased, so I could enlarge the context—and this larger context turned out to improve accuracy more than incorporating the lemmas did.

Surprisingly, it helped to consider not only continuous contexts, but also contexts with "holes", i.e. such that properties of a token are assigned by a rule, but the properties of a token between that token and the token whose parent is looked for are not assigned. However, it did not help to consider contexts where the position of a token with given properties was not fixed, e.g. when a rule stated that a token with tag `N4` had to occur somewhere between the first and the fifth position to the right from the token whose parent was looked for. A possible explanation of this phenomenon is that there are situations in which we do not need to know about all nodes in a context in case we know about those which can affect the dependency edge in question. For example, the rule `tag_0=Z: tag_1=P1 tag_3=Vp => par=+1Vp` was generated saying that a punctuation mark which is followed by a relative pronoun in nominative, by any token, and by a verb in past tense should have this verb as its parent. The situation corresponds to the beginning of a relative clause and, according to the training data, the undetermined token can be an adverb, a verb in the conditional form, a noun or a pronoun—none of these possibilities can alter the dependency between the punctuation mark and the verb.

After numerous experiments with various rule templates I achieved the best results with the templates which use only the morphological tags and which are designed this way: the rule templates cover all the rules with a continuous context of length up to seven containing the current position and, moreover, they cover all the rules containing the current position and up to two positions in the maximum distance of five. The accuracy is 74.5%.

The list of twelve of the most useful rules is shown in Table 2.2.[11]

[10] Memory requirements of the training phase were the limiting factor during almost all my computations.

[11] These rules are generated, indeed, with initialization of training data to the umbrella shape, to have the whole information contained in the rules.

```
tag_0=N6 => par=-1R6
tag_0=A2 => par=+1N2
tag_0=A1 => par=+1N1
tag_0=A4 => par=+1N4
tag_0=N2 => par=-1R2
tag_0=N4 => par=-1R4
tag_0=Z: tag_1=J, => par=+1J,
tag_0=A6 => par=+1N6
tag_0=N2 tag_-1=N1 => par=-1N1
tag_0=Vf => par=-1VB
tag_0=Db => par=+1VB
tag_0=P4 => par=+1VB
```

Table 2.2: The best rules for Method 2

I made several experiments with settings of some options of *fnTBL* and found out that it was better to set `ORDER_BASED_ON_SIZE=1`. The `threshold` option should have its default value.

2.4.4 Relocation of nodes hanging on the root

When rules assign a non-root parent to a node, but the node was dangling or it was detached of its parent because of cycles in the structure (see Section 2.4.2) and there are no other rules proposing a suitable parent, the node gets, as the last resort, the root of the sentence as its parent. We know almost for sure that the node is misplaced, but obviously we know little or nothing about its parent. There is, however, a special situation when we do know something: when such a node causes non-projectivity of an edge. Although non-projectivities are allowed in Czech, they are not frequent (according to [Zeman, 2005], there is about 1.9% of non-projective edges in PDT 1.0) and therefore, almost for sure, the node should be relocated to a position where it does not break projectivity. Even so, there are many possible parents, but it helps most to relocate the node so that it becomes a child of the parent node of the non-projective edge.

After the application of this procedure, the accuracy increased to 74.7% ($\Delta A = 0.2\%$), and this is the final accuracy of Method 2 on PDT 1.0.

When trained and tested on PDT 2.0 data, the final accuracy on d-test data is 74.8%. This and the previous figure can be compared to those in Table 2.1. For sake of completeness, the accuracy is 74.6% on the e-test data of PDT 2.0; and when the m-layer data with manual annotation were used, the accuracy rose

to 76.7% and 76.4% in the case of d-test and e-test data. The e-test data were used only for the purpose of this evaluation, which was performed only once.

2.4.5 Analysis of errors

All the statistics in this section are gathered on the d-test data of PDT 2.0 with their m-layer annotated manually.

In Table 2.3, statistics are presented in how many percent of cases the parser has assigned the correct parent to a node with the given two-letter m-tag. Moreover, in parentheses there is stated how many percent of child nodes have the given m-tag. For every part of speech exhibiting morphemic cases, records for all its cases are summarized into one number for the sake of lucidity and only one letter identifying this part of speech is then mentioned instead.

V	100.0	(0.0)	*AX*	78.0	(0.3)	*J,*	64.5	(1.9)
Vc	97.7	(0.4)	*Dg*	75.0	(1.7)	*VB*	63.2	(5.6)
RF	97.1	(0.0)	*Vf*	73.7	(1.9)	*J**	57.9	(0.0)
PX	96.2	(0.4)	*C=*	73.3	(1.7)	*RX*	57.1	(0.0)
A	94.0	(11.4)	*Z:*	73.1	(14.5)	*Vi*	56.8	(0.1)
P	87.1	(6.4)	*NX*	72.3	(1.7)	*J^*	53.0	(3.8)
N	82.8	(28.3)	*TT*	69.9	(0.5)	*Vs*	47.3	(0.7)
AC	82.7	(0.1)	*R*	67.5	(9.7)	*CX*	37.5	(0.0)
C	82.7	(1.2)	*Cv*	66.2	(0.1)	*Ve*	36.4	(0.0)
C}	80.4	(0.0)	*Vp*	66.1	(3.8)	*II*	10.0	(0.0)
Co	78.6	(0.0)	*Db*	65.6	(3.7)	*Vt*	0.0	(0.0)

Table 2.3: Precision of the assignment of parents

Similarly, Table 2.4 shows statistics of how many percent of cases the parser assigned a child correctly to a node with the given two-letter m-tag in. Strictly speaking: each figure (in percents) expresses the ratio of correctly proposed dependencies whose parent nodes have the given m-tag to all proposed dependencies of the same type; i.e. the precision of assignment is presented.

Figure 2.1 displays statistics of how many percent of cases an edge of a given length is correct in and how many percent of edges have a given length.

Figure 2.2 presents the distribution of distances between the assigned and the correct parent of a node, if the two differ. Misplacements concerning the root are treated as special cases which are not represented in the figure: 29.3% of misplaced nodes hang on the root; on the other hand, 3.9% of misplaced nodes should hang there.

CX	100.0	(0.0)	*Vf*	78.4	(2.3)	*Ve*	58.8	(0.0)
PX	100.0	(0.0)	*Vi*	76.1	(0.1)	*J**	57.9	(0.0)
R	92.0	(10.0)	*C=*	75.8	(1.2)	*Z:*	52.1	(2.0)
C}	91.7	(0.0)	*J^*	75.2	(6.9)	*Db*	51.4	(0.3)
AX	89.0	(0.1)	*Co*	75.0	(0.0)	*NX*	69.6	(1.0)
C	87.6	(0.4)	*TT*	74.1	(0.2)	*Cv*	65.8	(0.0)
N	84.4	(26.6)	*VB*	72.1	(15.0)	*Dg*	64.8	(0.4)
J,	82.9	(2.9)	*A*	72.0	(1.4)	root	60.8	(17.4)
Vp	82.3	(10.0)	*Vs*	71.7	(1.4)	*RX*	0.0	(0.0)
P	78.5	(0.3)	*AC*	70.1	(0.0)			

Table 2.4: Precision of the assignment of children

These statistics reveal no surprise: for the parser, the most difficult task is to place verbs (except for the conditional form of the verb *být* (*to be*), where the criteria are simple), adverbs, "prepositional phrases", coordinations (represented by coordinating conjunctions *J^*) and dependent clauses (represented by subordinating conjunctions *J,*). On the other hand, what could be called "noun phrases" in broader sense is usually correct: in most cases adjectives, pronouns and numerals expressed in words have the correct parent; and prepositions, nouns and numerals expressed in words have correctly assigned children. This conclusion corresponds to the statistics of accuracy of edges per their length: shorter edges, which probably correspond to "noun phrases", are more accurate.

2.4.6 Possible improvements

A possible way of improving accuracy can be to divide the training procedure into more phases. We train rules on one part of the training data and then we detect edges that are assigned in the most reliable way on the other part, based e.g. on m-tags of nodes constituting them or on their lengths (see Tables 2.3 and 2.4 and Figure 2.1). Then we create new data for training the unreliable edges only, since the reliable ones are retained from the first phase of training. The advantage is that in the second phase the training data can be extended with some features describing the partly created structure, e.g. informing about the number of children a node has after the first phase, and rules derived in this phase can profit from this additional and relatively reliable information.

It also may be worth to try to handle coordinations in a different way than until now, since coordinations, unlike the rest of edges, do not express dependency relations (see Section 1.2.2) and parsers are usually confused by this

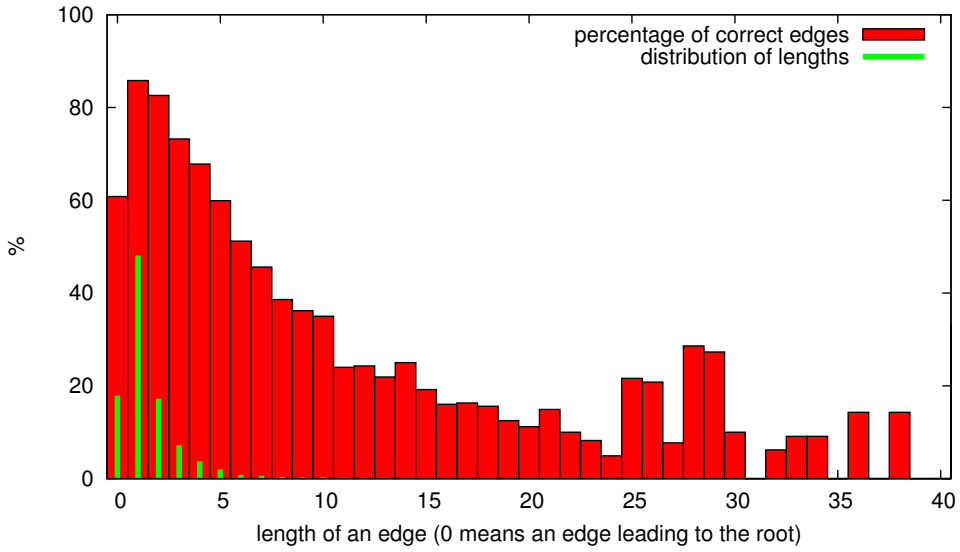


Figure 2.1: Percentage of correct edges with the given length and the distribution of lengths of edges

fact (only about a half of coordinating conjunctions were placed correctly, see Table 2.3). The idea is described below. The training data are altered in the following way: each node hangs on its effective parent instead of the direct one (the closer effective parent is used where there are more possibilities). Moreover, there is no need to generate rules assigning parents of coordination nodes: they hang on the root of the sentence. When the input data are parsed using rules obtained on these training data (see left side of Figure 2.3 for the example sentence), coordinations have to be taken into account again. First of all, the correct position of each coordination node has to be found: the "oldest" node such that the coordination node causes no non-projectivity when it becomes its child is chosen (see right side of Figure 2.3). Next, members of each coordination are assigned: they are the closest left and right siblings of the coordination node; they become children of the coordination node and their `is_member` attribute is set. After this step, if there is a comma (or other punctuation that can serve as the delimiter of members of the coordination) as the closest left sibling of the coordination node, it and its left sibling become children of the coordination node as well and the `is_member` attribute of the left sibling is set (not occurring in the example sentence); this step is repeated as many times as possible. The step of search for members of a coordination is repeated until there is a coordination node with no children (three times in the example sentence: for nodes *a*, –, and *a* taken from right to left; see left side of Figure 2.4). Using this approach, there are two possible sources of errors (beside those caused by mistakes in parsing): when there are nested

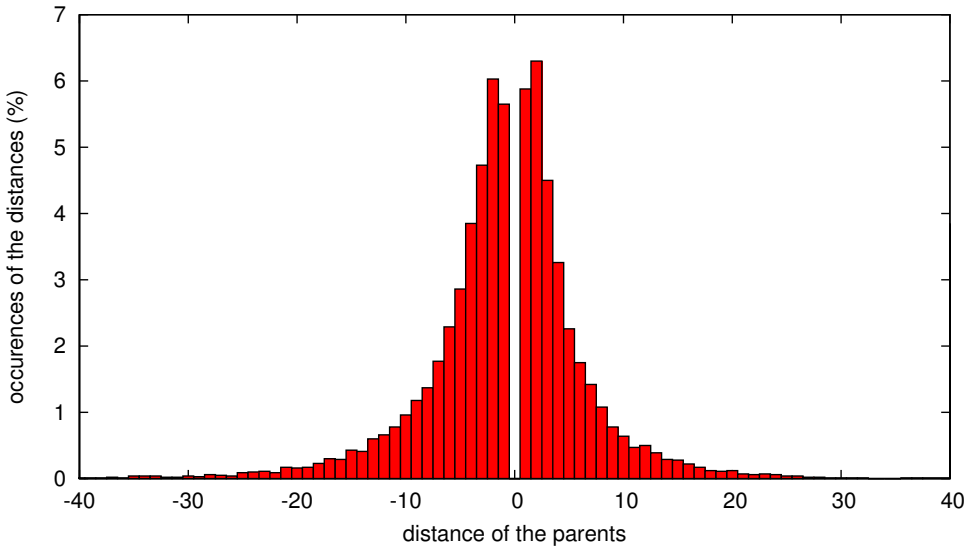


Figure 2.2: Percentage of non-zero distances between the assigned and the correct parent of a node

coordinations, it is impossible to determine which coordination node is the parent of the other one (in the example sentence, the error would originate if the coordination nodes were processed left-to-right); and, secondly, common modifications of members of a coordination are assigned as modifications of only one of them (see this problem when comparing to the correctly parsed sentence on the right side of Figure 2.4).

However, the problem of determination whether a modification is common for all members of a coordination, or whether it belongs to one of them only, seems to be relatively frequent source of errors of any parser. An alternative approach could help even in this case: this binary decision could be obtained by means of training using features of the modification, of its parent, of another member of the coordination, and possibly of the coordination node. This problem seems to be similar to the task of attachment of prepositional phrases in English.

The low number of non-projectivities in Czech can also be employed as a constraint helping improve the accuracy, when applied to parsed data. I described my first successful attempt in Section 2.4.4, but, obviously, much more can be done in this area. [Zeman, 2005] classified non-projectivities of Czech from the linguistic point of view, but his classification is preliminary, constraints based on it are hand-coded in his code and are specific for Czech, and he employed just a small part of the whole potential. It may be worth trying to infer the constraints by a method of machine learning and to use them automatically for

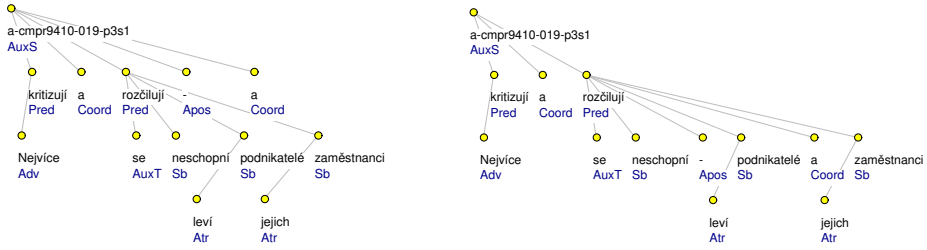


Figure 2.3: The sentence *Nejvíce kritizují a rozčilují se neschopní: leví podnikatelé a jejich zaměstnanci.* (*The most criticism and agitation comes from those incapable: incompetent entrepreneurs and their employees.*; lit.: *Most criticize and get-upset Refl those-incapable: incompetent entrepreneurs and their employees.*) parsed correctly according to the altered training data (left) and when its coordination nodes were adjusted (right). The s-tags are not important.

identifying and repairing non-projectivities, where the correct structures are projective. Note, however, that this task is almost reverse to that described in [Hall and Novák, 2005]—there the authors aimed to recover non-projectivities in places where they should be.[12] Their features seem to be reasonable even for the task described, but for this task I suggest their extending with the information whether the root of the subtree causing the non-projectivity of an edge depends on the root of the sentence and on the fact whether this root is an ancestor of the nodes forming the edge, or whether it is an offspring of their ancestor.

2.5 The comparison of the parsing methods

The accuracy of Method 2 is a bit higher than that of Method 1 (74.7% versus 73.6%); however, this difference is partly caused by different memory requirements of the methods. While Method 1 uses almost 2 GB of memory, Method 2 uses all the available memory, which is about 2.7 GB, and attempts to design rule templates which use approximately the same amount of memory as Method 1 result in the accuracy being several tenths of percent lower. Therefore, we can say that the performance of both methods is almost the same, although the methods are similar only to a certain extent. One possible explanation of this phenomenon is that only the features common for both methods are crucial for the results; those in which the methods differ are not substantial. If this is true, the accuracy is affected by the choice of features of

[12] Although not mentioned explicitly, their method is able to correct all local misplacements, not only those caused by improper handling of non-projective edges.

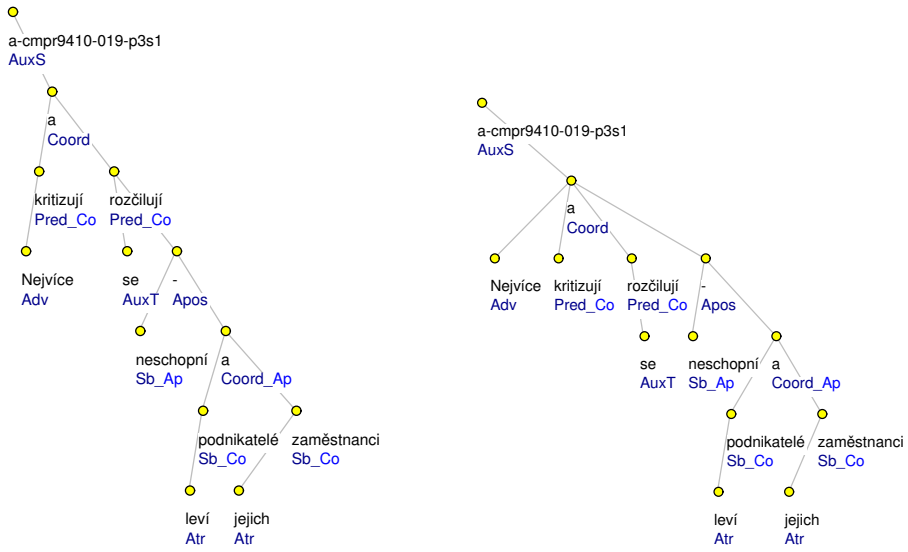


Figure 2.4: The sentence from Figure 2.3 after the whole procedure was performed (left) and correct annotation of the sentence (right).

tokens used in rules and by the choice to assign parents independently; it is not affected by the exact method of inference of rules and of parsing. From this it would follow that in order to achieve better accuracy, the former properties of the algorithm have to be improved.

2.6 Assignment of s-tags

To have a complete analytical structure as it exists in PDT, there is a need to assign an s-tag to each node yet. This task is much simpler than parsing, since it is a mere classification task, perfectly suitable for the *fnTBL* toolkit which I used for parsing.

2.6.1 Previous works

To my knowledge, there is the only tool assigning s-tags: it was created by Zdeněk Žabokrtský and was never published, although it was often used by annotators. He solves the same problem using the C4.5 tool, which performs classification by means of construction of decision trees. He uses the lemma (distinguishing only the 100 most frequent ones; the rest is replaced by the universal value meaning "other"), the part of speech, the detailed part of

speech, the case, and the voice of a node whose s-tag is being assigned, those of its parent, and, if its parent was not an autosemantic word, also those of its "youngest" ancestors being autosemantic words as the input features. He also uses a number of children of the node in question, distinguishing only values 0, 1, 2 and more. He reports the accuracy of his tool to be about 92.7%, when trained and tested on the appropriate data from PDT 1.0.

2.6.2 Design of rule templates

I extend his work and use *fnTBL* for the same task. I use the two-letter m-tag and the lemma of the node in question, its parent, its grandparent and its left and right tree siblings as the input features of the toolkit; I also use the number of children of the node in question with values 0, 1 and more. These 11 features are combined in the following way: a rule can contain up to four features and up to two of them can be lemmas, but when there are exactly four features, only one of them can be a lemma. Moreover, at least one of the features has to refer to the node in question.[13] The initial assignment of classes is done in such a way that the s-tag occurring most frequently in the training data is assigned to a node based on the combination of its two-letter m-tag and that of its parent.[14] Several of the most useful rules, only correcting the initial assignment, are given in Table 2.5.

I achieved the accuracy of 93.8% using the same data as Zdeněk Žabokrtský. I achieved 95.1% on the d-test data and 95.0% on the e-test data of PDT 2.0.[15] On the same data of PDT 2.0 annotated automatically at the m-layer and parsed by the parser described in [McDonald et al., 2005], I achieved the accuracy of 93.6% and 93.4%. The results reported were obtained for `ORDER_BASED_ON_SIZE=0` and `threshold=2`.

2.6.3 Attempt at an improvement: Two-phase training

Although the templates described above allow to use not only the features of the node in question but also those of its tree neighbours as far as we make them features of the node in question, it is impossible to use those features that

[13] These intricate templates resulted, as usual, as a trade-off between accuracy and memory requirements.

[14] Even this baseline assignment gives an accuracy of 81.7%.

[15] It should be said that in PDT 1.0, s-tags consist of two parts: the first part expresses what function the appropriate token has in a sentence; the second part expresses whether the token is a part of coordination and thus it supplies the attribute `is_member` missing in PDT 1.0. All the experiments of Zdeněk Žabokrtský and of mine are performed with this second part cut off, therefore, they are comparable. Up to now, we have no procedure for assignment of `is_member` at the a-layer, and thus the generated annotation is not complete.

```
tag=Z: child=2 => afun=Coord
lemma=" => afun=AuxG
tag=Jˆ child=0 => afun=AuxZ
lemma=, child=0 => afun=AuxX
lemma=) => afun=AuxG
tag=N6 ptag=R6 gtag=N1 => afun=Atr
lemma=( child=0 => afun=AuxG
tag=P4 lemma=on => afun=Obj
tag=N4 ptag=R4 gtag=N1 => afun=Atr
tag=N7 gtag=Vp => afun=Adv
tag=Vf plemma=být => afun=Sb
tag=N4 ptag=R4 gtag=N2 => afun=Atr
tag=N7 gtag=VB => afun=Adv
lemma=( child=2 => afun=Apos
tag=N1 ptag=Z: gtag=VB => afun=Sb
tag=Vp plemma=že => afun=Obj
```

Table 2.5: The most useful rules for s-tag assignment

are currently being assigned—only the features known before the training can be used. For our task this means that when assigning the s-tag of a node, we cannot use s-tags of its tree neighbours.

For this reason, I tried to split the training into two phases. In the first phase, rules were trained as written above, but only on a part of the data. Using these rules, s-tags were assigned to the rest of the training data. Now features of nodes in these data were extended with s-tags of its tree neighbours and based on these data, rules were trained to correct the assigned s-tags.

Several template sets were tested in the second phase, starting from those referring to s-tags only and ending with those using the whole set of information as the templates in the first phase plus s-tags. There were also two ratios which were the training data split according to. Although early experiments with this approach (using much simpler template sets in the first phase, and thus having a better chance to correct s-tags) improved the accuracy of about 0.3%, with the final template set, there was no improvement in the best cases.

Chapter 3

Tectogrammatical analysis

'It seems very pretty,' she said when she had finished it,
'but it's rather *hard to understand!' (You see she didn't*
like to confess, ever to herself, that she couldn't make it out at all.)
'Somehow it seems to fill my head with ideas—only I don't
exactly know what they are! However, somebody *killed* something*:*
that's clear, at any rate— '

[Carroll, 2003]

Development of a tool which is able to perform the **tectogrammatical analysis**, i.e. analysis which has annotation at the t-layer (described in Section 1.2.5) as its output, is a challenging work. Nevertheless, such a tool has a wide usage, as sketched in Section 1.1.

This chapter gives a description of the tool I developed for the mentioned task as well as an analysis of this task and a discussion of issues concerning the design of the tool. The tool needs an annotated corpus (see Section 1.2.2) in order to create its model (see Section 1.3) and tries to be independent on Czech (see Section 1.4).

In Section 3.1, the conditions in which my tool can be used are analyzed and also its limitations are given. Section 3.2 describes the evaluation method used for reporting results of both my tool and tools of other authors. Section 3.3 introduces existing works related to tectogrammatical analysis and it also gives baseline for my work. In Section 3.4 I describe tectogrammatical analysis from the technical point of view and introduce main ideas my tool is based on. Sections 3.5 to 3.8 describe particular phases of the operation of my tool. In Section 3.9, errors made by my tool are analyzed. Section 3.10 describes experiments with assignment of functors and with a *template reduction* mechanism.

Unless stated otherwise, the analyzer (and the tools described in the first part of Section 3.3) was trained on all the training data of PDT 2.0 annotated at the t-layer[1]; similarly, it was tested on all the d-test data of PDT 2.0 (prerelease version) annotated at the t-layer.

3.1 Scope and limits of the work

Annotation at the t-layer is a very complex task, this is why I had to find a compromise and focus only on several aspects of the annotation. Some attributes of the t-layer were omitted since they would probably not remain the same in the t-layer of languages other than Czech (this especially holds true for grammatemes); some were omitted because of extensive theory laying in their background: I would have to consider an enormous number of factors which affect values of these attributes, and it could complicate the development of my tool significantly. The latter is especially the case of the information structure.

On the other hand, there are several (partially overlapping) reasons why to implement the assignment of a certain attribute:

- it is necessary for the resulting data files to be technically tractable (e.g. `id`);
- it makes the core of the analysis (especially the structure);
- it is useful for easier assignment of other attributes (e.g. functor);
- not filling it would cause hardly recoverable loss of information (especially links to the a-layer);
- it is important for orientation during further work of annotators with preannotated data (e.g. `deepord`); or
- it is determined much more by the annotation scheme than by the language of a corpus (e.g. `is_generated`).

There are, however, two reasons for implementing the assignment of some attributes by a hand-coded procedure instead of machine learning. The first one is indicated in the last item of the list above: when an attribute is hopefully independent on the language of a corpus, rules for setting its value are always the same and it makes no sense to obtain them by training.

The second reason is that machine learning could be too strong and cumbersome a method for such a task. Instead of designing feature templates for the

[1] We should recall here that all the data of PDT 2.0 are human-annotated.

assignment of an attribute, modification of all (three) programs manipulating with data, re-creation of training data, retraining the rules and so on, a skillful programmer, having guidelines for annotation handy, could create a simple script implementing the respective rules for a certain language in a few hours with hypothetical 100% precision.

Thus, as already mentioned in Section 1.4, the attributes set by my tool are divided into two groups. In the first group there are those with values assigned based on training. Structure, functor, `t_lemma`, `nodetype`, `gram/sempos`, and `val_frame.rf` belong to this group. The tool thus relies on these attributes: this means that it supposes they exist and they have the same semantics as in PDT. It obviously does not rely on their particular values; the only exceptions are *complex* and *root* values of the `nodetype` attribute (the first one should exist and the second one, identifying the technical root of a sentence, should be set correctly for a data file to be valid) and *CONJ* and *APPS* values of the functor (they identify coordinations). This chapter describes the way of assigning the attributes from this group.

The attributes from the other group are being set by hand-coded procedures of the tool and their list and the way they are set by the tool follows.

- `a/aux.rf` and `a/lex.rf`: The procedure setting them is shown in Section 3.4.
- `atree.rf`: Set correctly, since no tree can originate nor can be deleted.
- `deepord`: Nodes originating from the a-layer are ordered according to their `ord` attribute, i.e. the same way they were ordered at the a-layer. For new nodes, it is set such that hopefully no non-projectivity arises, although this is not guaranteed.[2]
- `id`: It is set correctly, i.e. it has unique values. The shape of its values are similar to those in PDT.
- `is_generated`: Set correctly if it was correct to create the node.
- `is_member`: Its values adopted from the a-layer can be used in the first phase of processing only; then since the tree structure can change significantly, I developed a robust procedure setting this attribute based on tree structure and functors not considering previous values of the attribute.

As a consequence, the tool relies on these attributes as well as on their values.

Regarding the attributes from the m-layer, the tool relies on `tag`, including its particular positions, and on `lemma`, including special characters introducing its technical suffixes (not mentioned elsewhere in this text, see [Zeman

[2]The assignment of the attribute by a hand-coded procedure is a compromise: its assignment by training, which would be the best option, would be too labourious; on the other hand, the attribute really should be assigned in some way for the appearance of trees to be good.

et al., 2005]). As for the a-layer, the tool relies on `afun` and its values *Apos* and *Coord* (identifying appositions and coordinations) and *AuxK*, *AuxX*, and *AuxG* (denoting the final punctuation, a comma, and other graphical symbols—`id`s of these nodes are not written into `a/aux.rf` of a node if they are deleted, unlike in case of e.g. *AuxP* denoting prepositions). It relies on `ord` and `is_member` from this layer as well.

Thus, besides grammatemes and information structure, my tool does not deal with coreference, the rest of binary attributes capturing some properties of nodes (e.g. `is_name_of_person`), the `compl.rf` attribute capturing complement, the `quot` attribute expressing quotation, the `sentmod` attribute expressing sentence modality, and the `subfunctor` attribute capturing semantic variations of functors.

3.2 Evaluation

Comparison of t-trees is not as easy as that of a-trees, since it may happen by mistake that different tectogrammatical annotations of one sentence consist of different sets of nodes (see Section 1.2.5). Therefore, when we want to compare two t-trees in an attribute, so called alignment of nodes in t-trees has to be done at first, and then we can compare attributes of pairs of aligned nodes.

Formally, for N_1 and N_2 being the set of nodes of the first and second tree in comparison, respectively, we can define **alignment** as any simple[3] function $a : N_1 \mapsto N_2$. We call nodes $n_1 \in N_1, n_2 \in N_2$ **counterparts** of one another if $a(n_1) = n_2$. From this it follows that each node has at most one counterpart. If a node has a counterpart, we call this node **aligned** (and these two nodes **mutually aligned**), otherwise we call it **unaligned**.

Attributes of nodes can be thought of as a bunch of unary functions—one for each set of nodes; and each set is the domain of the respective function. For an attribute `attr` and sets N_1 and N_2, I will denote the respective function $\texttt{attr}_1$ and $\texttt{attr}_2$. To be able to represent a structure, we consider the parent of a node to be captured in virtual attribute `parent`.

We usually consider one of the sets of trees in comparison correct; let us call it the **golden annotation**; let us call the other set of trees the **test annotation**. This is also the reason why I write "this attribute of this node (in the test annotation) is correct" instead of more general "this node and its counterpart match in this attribute" etc. We denote the number of nodes in golden annotation as C_{gold} and that in the test annotation as C_{test}. We denote the number of pairs of mutually aligned nodes matching in the `attr` attribute as $C^{\texttt{attr}}$.

[3] I.e. $\forall n, n' \in N_1; a(n) \neq a(n')$.

When the attribute in question is not a reference to another node in the same set of nodes, the comparison is direct and we can define $C^{\texttt{attr}} = |\langle n_1, n_2\rangle \in N_1 \times N_2; a(n_1) = n_2 \wedge \texttt{attr}_1(n_1) = \texttt{attr}_2(n_2)|$.[4] Technical roots of sentences do not occur in any of these numbers.

However, in case of such a reference, $C^{\texttt{attr}}$ needs to be redefined because of different ranges of $\texttt{attr}_1$ and $\texttt{attr}_2$: the definition has to be changed to $C^{\texttt{attr}} = |\langle n_1, n_2\rangle \in N_1 \times N_2; a(n_1) = n_2 \wedge \texttt{attr}_2(a(n_1)) = a(\texttt{attr}_1(n_1))$. In the case of a parent, which is the only attribute of this sort we are interested in,[5] the definition then states that in order to be correct, the node and its parent should have their counterparts, and the counterpart of the parent of the node in question is the parent of the counterpart of the node in question.[6]

In both cases, I define **precision** $P_{\texttt{attr}}$ and **recall** $R_{\texttt{attr}}$ as

$$P_{\texttt{attr}} = \frac{C^{\texttt{attr}}}{C_{test}}, R_{\texttt{attr}} = \frac{C^{\texttt{attr}}}{C_{gold}}$$

To make one number of these two I define **F-measure** $F_{\texttt{attr}}$ in the usual way, i.e. as equally weighted harmonic mean of precision and recall:

$$F_{\texttt{attr}} = \frac{2P_{\texttt{attr}}R_{\texttt{attr}}}{P_{\texttt{attr}} + R_{\texttt{attr}}} = \frac{2C^{\texttt{attr}}}{C_{test} + C_{gold}}$$

There are, however, several attributes that require special treatment due to their nature. `a/aux.rf` is a set attribute—it holds unordered links to several nodes of the a-layer. In order not to complicate the evaluation, I decided to evaluate it *en bloc* and to consider this attribute to be correct only if the whole set of links matches with its counterpart from the golden annotation.

The value of the `deepord` attribute is a number expressing the left-to-right order of a node. Although it can be evaluated like another attributes, the evaluation would make little sense because of situations like the following one: when a sentence is (from the `deepord` point of view) correct except for the last node which is placed to the first position, `deepord`s of all the remaining nodes are shifted by one and all the nodes in the sentence are considered incorrect. For this reason I evaluate `deepord` as the agreement in relative order of counterparts of nodes neighbouring in the golden annotation. To be exact: $C^{\texttt{deepord}} = |\langle m, n\rangle \in N_1 \times N_1; m, n \text{ belong to the same sentence} \wedge \texttt{deepord}(n) - \texttt{deepord}(m) = 1 \wedge \exists a(m), a(n) \wedge a(m) < a(n)|$. Note that the number of such pairs of nodes can be at most less by one than number of its nodes in one sentence; that is why other counts have to be used while computing precision and recall, namely $C^{\texttt{deepord}}_{gold} = C_{gold} - \text{number of sentences}$ and analogically with $C^{\texttt{deepord}}_{test}$.

[4] Note that a node cannot be counted as correct in any of its attributes when it is unaligned.

[5] We do not deal with e.g. coreferences, which have the same nature.

[6] More informally, the "same" node in both trees has to depend on the "same" parent.

3.2.1 Alignment procedure

My procedure of tree alignment tries to mutually align the maximum number of nodes with the same a-link, and it endeavours to find such counterparts of the remaining nodes which match them in their positions or functors unless match in both of them is possible. The procedure is based on a comparison of structure, functors and a-links only and it is split into several steps.

1. From the trees in question, the nodes which have the same a-link (i.e. they originate from the a-layer) and which are the only nodes having this a-link are aligned. Typically, this step aligns the majority of nodes existing at the a-layer. This is the most obvious and most frequent case.

2. Only the nodes having an a-link are considered in this step, again, and such cases are solved when there is more than one node with the same a-link in one tree and there is at least one node with this a-link in the other tree. Therefore, this step typically tries to align the nodes which appear as copies of nodes existing at the a-layer, e.g. *malé a střední podniky* (*small and middle-sized companies*) at the a-layer is annotated as *malé podniky a střední podniky* (*small companies and middle-sized companies*) at the t-layer. The procedure of this step could be called **alignment by children**. The main idea is to mutually align two nodes having the same a-link and having the maximum number of their children mutually aligned. Formally, for each a-id aid and nodes $n_1 \in N_1; n_1 \notin \mathrm{Def}(a) \wedge \mathtt{alink}(n_1) = aid$ and $n_2 \in N_2; n_2 \notin \mathrm{Rng}(a) \wedge \mathtt{alink}(n_2) = aid$ I construct the following matrix:

   $$A_{n_1 n_2} = |n \in N_1; \mathtt{parent}_1(n) = n_1 \wedge \mathtt{parent}_2(a(n)) = n_2|$$

   Then I find nodes n_1 and n_2 for which $A_{n_1 n_2}$ is maximum, I mutually align them and delete the row and column corresponding to them from the matrix. I repeat this process until an unaligned node with the given a-link exists maximally in one of the trees—i.e. until the matrix is empty. This implies that nodes with the same a-link can be mutually aligned even if none of their children are mutually aligned.

3. For all yet unaligned nodes (even those with no a-link) an attempt is made to align them by children as in the previous step; the only differences are that their possible a-link is ignored and that at least one pair of the children of the nodes in question has to be aligned in order to align the nodes. In other words, we stop aligning when the matrix contains just zeros. Typically, inner nodes added at the t-layer are aligned in this step, e.g. nodes in ellipses.

4. Again, all yet unaligned nodes are being processed and an attempt is made to align them by parents and functors: those nodes which have the same functors and whose parents are mutually aligned are being mutually aligned in this step. Typically, (correctly assigned) valency members are aligned.

5. In this last step, all remaining unaligned nodes are being processed and an attempt is made to align them by parents *or* by functors—the nodes being mutually aligned should match in one of these attributes. Unfortunately, it is not optimal (in terms of aligning the maximum possible number of nodes) to align by parents at first and by functors afterwards, or vice versa. For example, for the situation in Figure 3.1, the alignment by functors at first and then by parents is optimal (nodes B and D and then A and C would be mutually aligned), but for the situation in Figure 3.2, this could result in the only, non-optimal alignment of nodes E and H. On the other hand, alignment by parents at first does not have to be optimal in the first situation (the only alignment of B and C), but it is optimal in the second one (E and G and then F and H). (The dotted line in the pictures indicates nodes mutually aligned.)

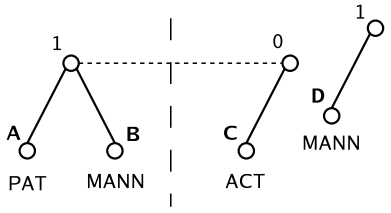


Figure 3.1: Alignment, problematic situation #1

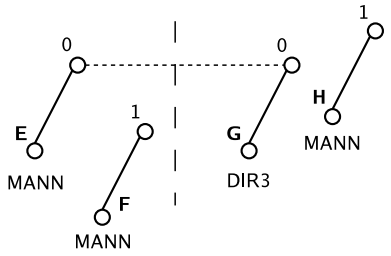


Figure 3.2: Alignment, problematic situation #2

For this reason I developed a procedure aligning by functors at first, which, however, takes the parents of nodes into account. For each aligned node, the number of its unaligned children which can be aligned

by functors is noted: it is equal to the number of its unaligned children minus the number of unaligned children of its counterpart if the node has more unaligned children than its counterpart; zero otherwise. (The numbers are stated in the figures by the respective nodes for the mentioned situations.) After that, nodes are being aligned by functors, but for a node at most the number of its children is aligned which is recorded along with the node. After that, nodes are being aligned by parents; from the above it is guaranteed that each node is aligned by parents if it could be aligned before.

In the mentioned situations, this alignment procedure is optimal.

3.3 Baseline and related work

Since the structures of the annotation of a sentence at the a-layer and the t-layer are similar to a certain extent, I take the t-layer annotation having the same nodes at the same places as in an a-layer annotation as the baseline of the structure annotation. For the purpose of the determining baseline for functor assignment, I take *RSTR* (restriction), the most frequent functor, to be the functor of all nodes. The result is then $P_{\texttt{parent}} = 50.3\%$, $R_{\texttt{parent}} = 62.1\%$, $F_{\texttt{parent}} = 55.6\%$ and $P_{\texttt{functor}} = 16.9\%$, $R_{\texttt{functor}} = 20.8\%$, $F_{\texttt{functor}} = 18.6\%$.

There are three tools performing partial annotation of (mainly) tectogrammatical structure and functors available. They all were developed to reduce manual work needed to annotate PDT. The first tool used in the annotation process, preannotating at the t-layer based on manual a-layer annotation, was *AR2TR* ([Böhmová, 2001]). Its algorithm is hand-written and performs the following tasks:

- assigns the value of functors in clear-cut cases,
- assigns the values of several attributes (e.g. verbal and sentence modalities and aspect),
- deletes most nodes of synsemantic words and fills the corresponding attributes of their parents accordingly, and
- relocates nodes in certain cases and thus adjusts the tree structure.

The output of *AR2TR* was taken as the input of decision-tree-based tool for assignment of functors called *AFAS* ([Sgall et al., 2002]). It was intended to be used once the tectogrammatical structure is correctly assigned; annotators reported that it made their work easier even in this phase, though. *AFAS* makes decisions based on five types of information about nodes: lexical (lemmas),

morphological (several positions of m-tags), analytical (s-tags), topological (numbers of children), and ontological (positions of lemmas in the EuroWordNet Top Ontology). When applied to tectogrammatical structures assigned manually, excluding the nodes that do not originate from the a-layer, the authors report the accuracy of functors 82.3%, having the data consisting of mere 27,463 nodes for training and testing purposes.

After applying those two tools on the baseline of the test data, the result is $P_{\texttt{parent}} = 87.0\%$, $R_{\texttt{parent}} = 78.8\%$, $F_{\texttt{parent}} = 82.7\%$ and $P_{\texttt{functor}} = 72.7\%$, $R_{\texttt{functor}} = 65.8\%$, $F_{\texttt{functor}} = 69.1\%$.[7]

The third tool called *AddVal* ([Hajič and Honetschläger, 2003], [Honetschläger, 2003]), which uses a valency lexicon for adding nodes and correcting functors, was developed by me near the end of the main annotation process of PDT 2.0 and thus was not used broadly by annotators; however, it was used in a project of machine translation ([Hajič et al., 2003]). When applied to the output of the previous tools, the result improved to $P_{\texttt{parent}} = 86.4\%$, $R_{\texttt{parent}} = 82.9\%$, $F_{\texttt{parent}} = 84.6\%$ and $P_{\texttt{functor}} = 74.1\%$, $R_{\texttt{functor}} = 71.1\%$, $F_{\texttt{functor}} = 72.6\%$.

A question can be asked whether I can use the output of these tools as the starting point of my analyzer. The answer may not be obvious, although it is definite: I cannot. The tools were developed for Czech, while my analyzer is intended to work regardless of the used language—while *AFAS* can be retrained and *AddVal* can use the appropriate valency lexicon, *AR2TR* profits from properties of Czech heavily and cannot be used for other language. However, even for Czech, these tools cannot be used for preprocessing any more: they (mainly *AR2TR*, again) can process only data in an older format than PML is in—the one used in previous versions of PDT. Unfortunately, the conversion from this format into PML cannot be fully automated: because of some deficiencies and ambiguities of the older format, the conversion always requires manual work. This is why the input of my analyzer is always the data annotated at the a-layer.

Besides the mentioned ones, there are other interesting works available regarding subtasks of annotation at the t-layer, as the PDT has it, or at a conceptually similar layer coming from another theories of language description. The following overview can serve as an illustration of what tasks are manageable by a computer; for me, it also served as an inspiration whenever I needed to solve the respective tasks. I did not compare results of these works to those of mine.

In [Bojar, 2003], the author aims at extraction of so called subcategorization frames based on frames of verbs observed in a corpus. Moreover, he discusses

[7] Needless to say that if *AFAS* was retrained using the whole data (which did not exist when it was under development), we could expect better performance.

issues of assigning the valency frames based on subcategorization frames and states that the assignment of functors, which considers the basic step of the procedure, should be based on the morphemic realization of a modifier and on lemmas of both the modifier and its governing verb. He documents this statement by statistics and examples from the corpus.

Authors of [Lopatková et al., 2005] perform word sense disambiguation, which is in fact valency frame disambiguation, since each frame in the valency lexicon they use corresponds to one meaning of a word. For this task they use the C5 toolkit implementing the decision-tree learning. They report that the best set of features that the decisions are based on is the one consisting mainly of binary values expressing the presence of a dependent having certain morphological and/or syntactical characteristics concerning e.g. its case or its preposition; in short, they check for certain morphological realizations of dependents.

Authors of [Razímová and Žabokrtský, 2005] did an admirable job regarding grammatemes. For PDT 2.0, they designed a system of grammatemes which arises from a hierarchical typology of tectogrammatical nodes which they introduced as well. Then, they hand-wrote rules assigning values to all grammatemes as well as to hierarchical attributes of nodes; they used lexical lists extracted from PDT for some subtasks. The annotation made by their procedure became part of the tectogrammatical preannotation of data of PDT 2.0.

In [Kučová and Žabokrtský, 2005], the scheme of annotation of coreferences in PDT is introduced and an algorithm for annotating textual coreferences of personal pronouns (i.e. for the task of anaphora resolution) is proposed. The authors are able to achieve accuracy 60.4% in this task by means of hand-written rules, which is comparable to results of automatic anaphora resolution in English.

Although work of [Postolache, 2005] suffers from certain drawbacks in presentation and interpretation of results, the author definitely shows that the assignment of information structure is, to a large extent, a task manageable by means of machine learning. When she confined herself to the topic/focus distinction of nouns and pronouns in PDT 2.0, she reached F-measure of 82.0%, which is far above the performance of her baseline assignment.

The tectogrammatical relation of a pair of nodes, which is captured in functors in PDT, is usually called "semantic role" in other theories and there are many articles focusing on the issue of automatic assignment of this information. However, the articles are mostly aimed at English with its traditional constituency description[8] and since features used there relate to constituency

[8]The methods used are often traditional as well, i.e. tagging a token as being inside/outside/at the beginning of a phrase corresponding to a certain semantic role.

structures mainly, the leading ideas of the articles can be hardly used in a dependency approach. One of the exceptions is the article [Hacioglu, 2004], where the author converts parsed English texts into dependency trees before the assignment. The features the author uses are lexical values of the dependent and the governor nodes, their mutual position, their morphological tags and some other features concerning the structure of the sentence. He used support vector machines for the classification.

The author of [Semecký, 2006] deals with the automatic disambiguation of verb valency frames.[9] He performs this task by means of a decision tree toolkit using a corpus automatically annotated at the morphological and syntactical layer. The features of the verb and its context that he used can be grouped to five bunches: morphological ones, those based on syntax (morphological and lexical characteristics of words dependent on the verb), those testing occurrences of idiomatic expressions, animacy of dependent words, and information from WordNet top-ontology classes. The author reports about 41% error reduction on the baseline consisting of assigning the most probable frame to each verb.

3.4 Introduction to the method

From the point of view of structure, which is the basis of tectogrammatical annotation, changes between the a-layer and t-layer can be technically described as:

- deletion of nodes (synsemantic words or majority of punctuation)
- relocation of nodes (rhematizers; phrases having different structure at the a-layer and the t-layer etc.)
- copying of nodes (filling the ellipses like *Zákazníci budou platit o 10 % víc, někteří [zákazníci budou platit] ještě víc* (*Consumers will pay 10% more, some [consumers will pay] even more.*))
- creation of nodes (filling the valency slots in the vast majority of cases; and filling ellipses like *[] Tabulka 1* (*[] Table 1*))

The *fnTBL* toolkit, which I use for the tectogrammatical annotation, is not capable of processing tree structures. Therefore, when we want to pass features of the parent, children etc. of a node to the toolkit, we have to pass them as

[9] Because of the way the used valency lexicon is constructed, this actually is word sense disambiguation.

if they were the features of the node. The biggest disadvantage of this procedure is that the toolkit cannot employ the tree structure in its current state—it remains the same during the whole training or classification process and can be modified as late as the process is finished. For example, the toolkit modified the functor of the parent of a node during training; however, this change cannot be reflected in rules generated consecutively, because this functor is written as a feature of the mentioned node.

In order to evade the mentioned disadvantage, I split the training (and thus, inevitably, the classification as well) process into several phases. After each phase of training, a set of trees being processed is modified according to rules created in this phase; and these are the modified trees which enter the next phase. The ideas (some of which I got after exploring of data) leading to the division follow.

- When we want to fill valency slots, it is necessary to know which valency modifications are already present. We need to have functors already assigned and all existing nodes at their places in order to do this.

- When a node having children is being deleted, its children have to be relocated. In the vast majority of cases, one of the children takes place of the deleted node and becomes the new parent of its siblings. We will call such child a **successor** of its parent. Moreover, the `a/aux.rf` attribute of the successor is populated with links to the a-layer counterparts of its deleted parent. Therefore, if I assigned the successor and performed the operations mentioned above, it would not be necessary to generate many complex (and thus unreliable) rules performing the relocation of children. From this it follows that the deletion should be done before the relocation.

- When assigning functors, it is better not to have deleted nodes that correspond to synsemantic words, since they influence the functors a lot.

- Obviously, when assigning the attributes of a node, the node has to exist.

Having considered these requirements I split the process into four phases:

1. deletion of nodes and assignment of functors;

2. relocating and copying the nodes together with creation of inner nodes (i.e. of those added because of ellipses); functors of new (i.e. copied or newly created) nodes are also assigned here;

3. creation of leaf nodes (i.e. of those corresponding to valency modifications) and assignment of their functors as well as assigning the `val_frame.rf` attribute[10];

4. assignment of the rest of the attributes.

This sequence has another advantages as well: since functors are assigned at the very beginning, they might significantly help in the second, most complicated phase. Moreover, synsemantic words, coming handy almost only for the assignment of functors, are deleted at the beginning[11]; and valency modifications, useless in the parsing process, are created as late as the rest of the structure is assigned—both facts contribute to the clarity of the "battle field".

For each of these phases, three programs have to be created: one comparing input data and desired output data, both in PML format, and creating training data for *fnTBL*; another one converting data from the PML format into the format required by *fnTBL* classification; and one modifying the data in the PML format according to the output of *fnTBL* classification. The second of them is simple enough; how the other two work is stated in the description of respective phases.

From the short description of the phases given above we can guess that rule templates optimal for each of them will be completely different. If we had a toolkit capable of processing tree structures, it would probably be possible to give it the union of all four rule templates and perform the training in one phase; however, huge memory requirements (limiting even in the four-phase training) would probably force us to split the training into several phases, which would consume less memory. From this point of view, the incapability of the *fnTBL* toolkit to process trees is not such a disadvantage, as it might seem at the first glance. A disadvantage that remains is that in case of four-phase training by a toolkit capable of processing trees, programs converting the data would be less complicated than in case of *fnTBL*; on the other hand, their number would be the same.

There are even more disadvantages (depending on the described one) of the usage of the *fnTBL* toolkit for tectogrammatical parsing. One of them is multiplication of information: e.g. rules may assign X as the functor of the parent of node A and Y as the functor of the parent of node B—but what to do if A and B are siblings? Furthermore, we need to handle the variable number of children of a node and consequently the variable number of their features. If the toolkit is able to process tree structures, it would probably cope with this.

[10] Nodes created in this phase have no valency frame, so there is no need to wait for the last phase, and features used for deciding about valency modifications are, naturally, good for making decisions of the valency frames as well.

[11] Whenever needed, they can be reached through the `a/aux.rf` attribute.

How these disadvantages are evaded in particular phases of the parsing (if they occur there), is stated in the description of the phases.

On the other hand, it should be noted that most of general tools for machine learning can process only classification tasks and use of one of them on manipulation with trees would probably bring very similar difficulties.

3.5 Phase 1: Deletion of synsemantic nodes and functor assignment

The aim of this phase of tectogrammatical parsing is to delete nodes that do not occur at the t-layer any more and to assign functors to the remaining ones.

3.5.1 Design of the procedure

Assigning functors to nodes is an elementary classification task. We can simply perform the deletion of nodes along with it: when a node should be deleted, a special value of the functor meaning "deleted" could be assigned to it.[12] However, not only a node should be deleted, but also its successor should be appointed from among its children if the node has any (see Section 3.4). Here, one disadvantage of using *fnTBL* for this task turns up: we would like to employ some features of the node in question and of its child or children; however, manipulation with the variable number of children will be inconvenient at least. I evaded this by extending the rule templates with another type of rule templates, deleting the *parent* of a node together with appointing this node as the successor of its parent. We can see that both types of rule templates have to exist, since the second type is not able to delete leaf nodes.

A node is being deleted if a single rule of any type states so. If the node has children and its successor is not appointed, it is deleted only if it has just one child (which is not planned to be deleted as well)—and this child becomes its successor. The node is retained otherwise.[13]

[12] One could object that this task should have been divided into two—the first for making a decision of whether a node should be deleted and the second for assigning the functors of nodes not deleted—since the decision of deletion is more important than the assignment of functors and the classification into mere two classes would be more accurate. However, from the confusion matrix corresponding to the classification I performed it follows that the classification into the "deleted" class is the most reliable one.

[13] In the test data, a node was retained by this mechanism in 0.4% of cases when it should have been deleted.

3.5.2 Design of rule templates

For both types of rules, I used only features of the node in question and its parent.[14] Namely I worked with their lemmas (*lemma*), s-tags (*afun*) and separate positions of their m-tags except for both possessive subtags, the subtag of variant, and both reserved positions, which were obviously useless. This makes up 24 features, too much to fit into memory even for rules containing a low number of features: if rule templates can contain maximum of three features with the condition that when there are exactly three features, they have to belong to different nodes, the number of templates is as high as 3,768. I did not want to prune the templates manually, since I supposed that it would hurt the accuracy.[15]

In order to solve this problem, I developed an automated adaptive procedure reducing the number of templates. It operates in the following way: all templates are used for training the rules on a small part of data, which can still fit into memory. From the resulting rule file, the most useful templates are extracted into a new template file. Strictly speaking, each template with the number of instances (i.e. rules inferred from it) is higher than a certain threshold is retained. The rule file is then completely discarded and the new set of templates is used for training using the whole data. I call this technique **template reduction**. In Section 3.10.1, several of its properties are shown.

In this case, I used the template reduction with the threshold of two rules per template, and mere 106 templates remained.

Handling situations when a coordination should be a successor of a node needs special care, since we need to satisfy two requirements of the annotation scheme: it is a coordination node what adopts other children of its deleted father (however rare this situation may be); on the other hand, they are members of the coordination whose `a/aux.rf` attributes are populated with a link to the deleted node. Moreover, we need to be as consistent as possible in order not to confuse the rules. For these reasons, the training data looks as follows:

- For nodes being members of a coordination, features of their effective parent are considered, rather than those of their direct parent. (This holds true for the testing data as well.)
- Members of a coordination are marked as successors.
- A coordination node is marked as a successor as well if all members of the coordination contain link to that parent in their `a/aux.rf` attributes.

[14] Memory was the limiting factor during all computations.
[15] As stated below, this assumption proved true.

`afun=Atr => func=RSTR`
`spos=: => func=deleted` (punctuation)
`afun=AuxP => func=deleted` (prepositions)
`pos=N par_afun=AuxP => par_del=yes` (a noun with a preposition)
`afun=Sb => func=ACT`
`afun=Pred case=- => func=PRED`
`afun=Obj => func=PAT`
`afun=Coord case=- => func=CONJ`
`afun=Adv => func=TWHEN`
`spos=, => func=deleted` (subordinating conjunctions)
`lemma=se spos=7 => func=deleted` (reflexive pronoun *se*)
`afun=AuxV case=- => func=deleted` (auxiliary verb *být* (*to be*))

Table 3.1: The best rules in the first phase

3.5.3 Experiments, results and statistics

The described set of rule templates proved to be the best, giving the precision, recall and F-measure of the structure assignment $P_{\texttt{parent}} = 87.1\%$, $R_{\texttt{parent}} = 76.2\%$, $F_{\texttt{parent}} = 81.3\%$ and those of the functor assignment $P_{\texttt{functor}} = 85.9\%$, $R_{\texttt{functor}} = 75.2\%$, $F_{\texttt{functor}} = 80.2\%$. Both "output" features of a node were initialized based on s-tags of the node and of its parent.

Twelve of the most useful rules[16] are shown in Table 3.1.[17]

In Figure 3.3, there is an example sentence correctly annotated at the a-layer and tectogrammatically annotated after Phase 1. The annotation is correct except for nodes *psychologický* (its correct functor is *REG*) and *hodně* (correctly *LOC*).

When I tried to prefer the features which I regarded as more important (lemma, s-tag, and part of speech) in templates, striving to reduce the number of templates to bundle them into memory, results were at the best the same as described above. Template reduction proved to be the best option.

I tried to incorporate the functor of a node in question (being assigned at the moment) into features. Since this is not a new information, I expected the results not to change, but the number of rules to decrease. However, not only results, but also number of rules were almost the same.

[16] *pos* denotes part of speech, *spos* denotes detailed part of speech, *par* prefix denotes a feature of the parent.

[17] These are, indeed, for `ORDER_BASED_ON_SIZE=1` so that the rules do not contain redundant atomic predicates and thus are more readable for a human; and without initialization of the training data, so that the whole information is contained in the rules.

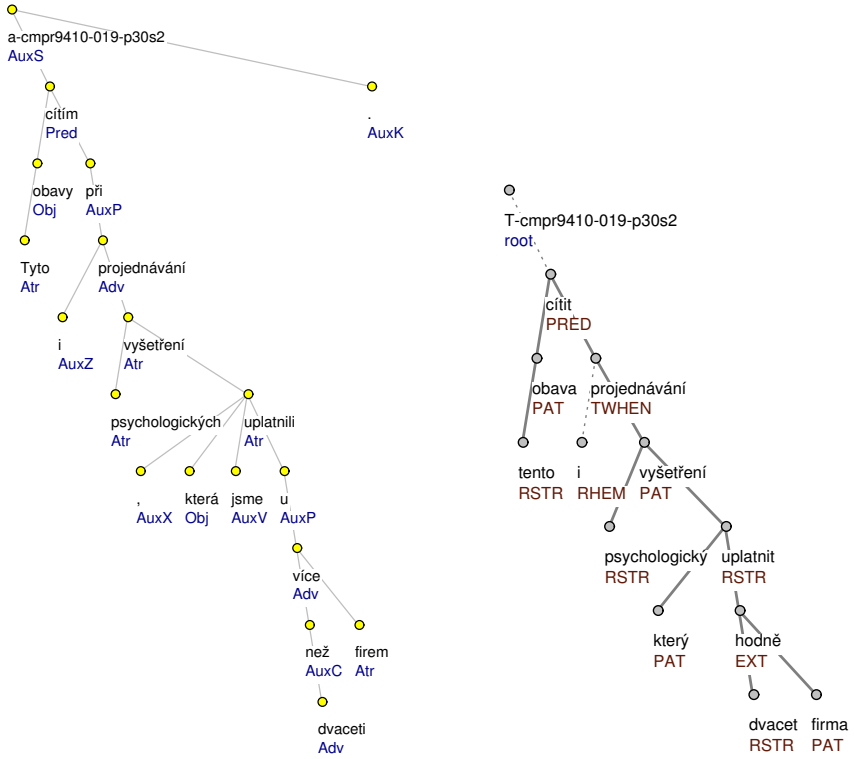


Figure 3.3: The sentence *Tyto obavy cítím i při projednávání psychologických vyšetření, která jsme uplatnili u více než dvaceti firem.* (*I feel these concerns also in discussions of psychological examinations that we applied with more than twenty companies.*; lit.: *These concerns I-feel also in discussions of-psychological examinations that we-are applied by more than twenty companies.*) at the a-layer (left) and its tectogrammatical annotation after Phase 1 (right)

I also experimented with reducing the number of templates by reducing the number of features using a mechanism similar to template reduction. I generated all templates with at most three of all 24 features and trained them on the largest data which were able to fit into memory—thus on very small data. Then I chose 12 features which took part in the assignment of properties of the highest number of nodes and generated all templates with at most four features out of those twelve. As late as at this moment I performed the template reduction for them etc. Although the set of chosen features looked very reasonably, the result was worse ($\Delta F_{\texttt{parent}} = -0.7\%, \Delta F_{\texttt{functor}} = -0.6\%$).

I also tested the performance of two-letter m-tags (described in 1.2.3). I took the lemma, afun, and two-letter m-tag of a node in question and of its parent, having mere six features. I generated all possible templates of them and per-

formed training with this full set of templates without a need of performing template reduction. Even with this one-step training I obtained the result just a little worse than the best one ($\Delta F_{\texttt{parent}} = -0.3\%, \Delta F_{\texttt{functor}} = -0.3\%$).

I experimented with several parameters of the *fnTBL* toolkit, namely `ORDER_BASED_ON_SIZE` and `threshold` and I found that their various settings had only insignificant impact on the result in this phase. The results reported were obtained for `ORDER_BASED_ON_SIZE=0` and `threshold=1`.

3.6 Phase 2: General transformations

The aim of this phase of parsing is relocation and copying of nodes and creation of inner nodes. Functors of new nodes should also be assigned.

3.6.1 Analysis of the problem

The transformations this phase should perform are considerably complex. Several examples of fragments of sentences from the data—where a node should be copied within coordinations only—document it. In Figure 3.4, the situation is depicted where the word having its modifications coordinated should be copied. On the contrary, Figure 3.5 shows a common modification of the coordinated words. In Figure 3.6, yet another word should be copied because of an ellipsis in the coordination.

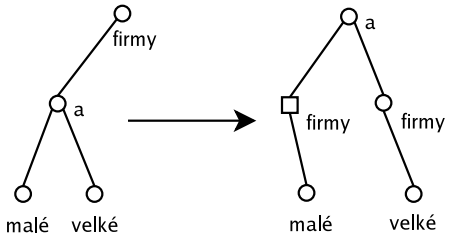


Figure 3.4: Correct annotation of *malé [firmy] a velké firmy* (*small [companies] and middle-sized companies*) at the a-layer and t-layer; newly added nodes are rendered as squares

If I describe such transformation as a composition of simple actions, e.g. of copying a node and relocation of a node[18], and perform the training in this

[18]Then, the transformation in Figure 3.4 cannot be described as composition of less than five actions.

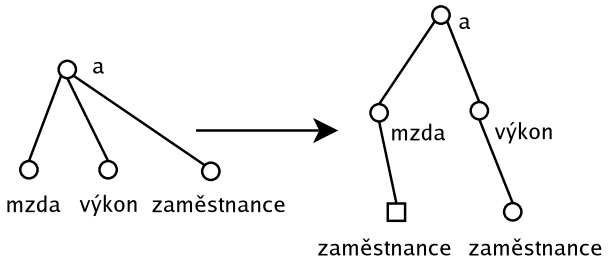


Figure 3.5: Correct annotation of *mzda [zaměstnance] a výkon zaměstnance* (*wages [of an employee] and performance of an employee*; lit.: *wages [of-an-employee] and performance of-an-employee*) at the a-layer and t-layer

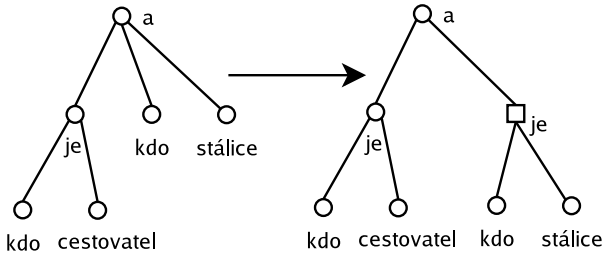


Figure 3.6: Correct annotation of *kdo je cestovatel a kdo [je] stálice* (*who is a traveller and who [is] a fixed star*; lit.: *who is a-traveller and who [is] a-fixed-star*) at the a-layer and t-layer

manner, then during parsing, only some of these actions may be performed, since I do not have any means available to ensure that either all actions, or none of them are performed.[19] For this reason and because of the fact that "reasonable" transformations should occur in the training data quite often[20], I decided to describe each transformation with a single formula.

3.6.2 Design of the procedure

I describe a transformation as a subtree-to-subtree transformation in the following manner. The root of the subtree being transformed is denoted `A` and

[19] In addition, this is complicated by the usage of *fnTBL*, because the processed tree remains effectively unchanged during processing, and thus actions have to be described as if they have to be performed on the initial state of a tree, not on the state which is step by step coming closer to the desired state.

[20] The transformations from Figures 3.4, 3.5, and 3.6, occur in their pure forms (i.e. not being a part of an even more complex transformation) 178 times, 14 times, and 81 times, respectively, in the training data. Even the smallest of these counts is sufficient for a TBL method.

```
A(B(C))->A(B,C)
A(B)->B(A)
A(B,C)->A(B(C))
A(B)->A(*(B))
A(B)->*(A,B)
A(B,C)->A(C(B))
A(B,C)->A(*(B,C))
A(B)->B
A(B)->A
A(B(C,D))->B(A(D),a(C))
A(B,C)->C(A,B)
A(B,C)->*(A,B,C)
```

Table 3.2: The best transformations in the second phase

other nodes occurring in the transformation are denoted by the following letters in the depth-first order. The children of a node are stated behind it in parentheses according to the left-to-right order they have in the tree and they are separated with commas. In the resulting tree, each node is denoted by the same letter as it was assigned in the original tree (if it still exists: because it took no extra effort, I implemented deletion of nodes as well—at least, nodes being retained by mistake in the previous phase can be deleted in this phase). New nodes are marked with `*` and a copy of a node is marked with the lowercase variant of the respective letter. In the description of the resulting tree, siblings are sorted in alphabetical order.[21] The transformation is the smallest possible, i.e. besides nodes being created, deleted, copied, or moved, only nodes required in order for the original and resulting structures to be trees are involved. Moreover, nodes which do not exist in the original tree and which are leaves in the resulting tree are not involved—this is the task of the next phase.

With this knowledge, transformations from Figures 3.4 to 3.6 can be described as `A(B(C,D))->B(a(C),A(D))`, `A(B,C,D)->A(B(d),C(D))`, and `A(B,C,D)->A(B,b(C,D))`.[22]

Twelve transformations occurring most frequently in the training data are shown in Table 3.2.

Several problems remain to be solved. One of them is to determine which node a transformation (involving several nodes) should be stated along. Every transformation involves at least two nodes: the root of a subtree modified by

[21] As this will also be their actual left-to-right order in the tree, it may be incorrect.

[22] For the reason stated in the previous paragraph, nodes *kdo* (*who*) (the first one) and *cestovatel* (*traveller*) are not involved in the last of the transformations.

the transformation and one of its children. Of these two, I chose the latter one (if more children of the root were involved, I chose the leftmost one), mainly for the reason that its parent could be referenced easily.

There is also a need to identify the nodes involved in a transformation. The above implies that the node by which a transformation is stated always occurs as `B` in the transformation and the parent of the node is `A`, but for other possible nodes, we know only their mutual positions. This is usually insufficient for their identification. Therefore I decided to attach certain characteristics of `C` and the following nodes to the transcription of a transformation in training data. For this task, s-tags proved to be the best option (see 3.6.4). Thus, when a transformation should be performed, not only that relations among nodes stated on the left side of the formula have to be fulfilled, but also all nodes have to have the given characteristics—otherwise, the transformation is not realized.

When a node is created or copied in this phase, its functor should be assigned. Since the creation or copying of a node is a part of a transformation, it seems natural for the functor assignment to be part of it as well. I solve it again by attaching the functors of new and copied nodes to the transcription of a transformation. The functor of a copied node can also have a special value meaning "the same as the original node has".[23]

When a node is being deleted,[24] its successor should be appointed somehow, if the node has children. As before, I attach certain characteristics of the successor to the transcription of a transformation, namely the character belonging to the successor. The character can have two special values, one meaning "no successor" and another one meaning "successor outside the transcription of the transformation".[25]

During the preparation of training data for *fnTBL*, I had to solve some other problems: what to do with transformations which should create a node with its a-link pointing into another sentence; or where a node which was deleted by mistake in the previous phase should occur. Since transformations, as I proposed them, are restricted to one sentence only, the first problem can be solved just partially: a node can be created, but its a-link cannot be set correctly. As for the other problem, I have no reasonable proposal how to re-establish a deleted node. Having considered the fact that these problems are relatively

[23] This is the most frequent case.

[24] In a transformation, this is given implicitly: it occurs on the left side of the transformation, but not on its right side.

[25] In the latter case, the (first) child of the root of the sentence is considered to be the successor. It would be probably better to design the transcription of a transformation to always contain successors.

rare, I decided not to solve them at all and to generate no transformation in these cases.[26]

The way of creating a transcription is illustrated in an example of a (real) complex transformation. The transformation `A(B(C(D),E(F,G)))->A(B(D(C),a(F,G,b)))+Adv+Atr+AuxC+ExD+ExD;CPR+@|A` can be applied only if the s-tag of node `C` is *Adv*, the one of node `D` is *Atr* etc. (given after the last parenthesis). The transformation sets the functor of node `a`, which originated as the copy of node `A`, to *CPR*, and it sets the functor of node `b` to the value the node `B` has (given after the semicolon). Similarly, if a new node occurred in the transformation (it would have been denoted with an asterisk), its functor would be attached (given after a colon). The deleted node `E` has the node `A` as its successor (given after the `|` symbol)—if `E` has some other children than `F` and `G`, those children become children of `A`.

3.6.3 Design of rule templates

The idea that *fnTBL* making decisions about the performance of a transformation should have access to features of all the nodes involved in the transformation sounds natural. There is, however, the following matter to take into account: the number of such nodes is variable and can be very high. Although each of the twelve most frequent transformations given above involves four nodes at maximum, there are transformations involving over 25 nodes, which is the limit I set for the transformation be plausible. This problem can be solved in several ways:

- Guaranteeing that even the largest transformation found in the training data has access to features of all nodes involved in it. This leads to the situation that together with every node there features of typically all nodes in its subtree (and of its parent) have to be recorded. The problem is not the sparseness of such data, but the number of features which cannot fit into memory.

- Creating individual training data for every transformation. Then, classification is a binary decision whether the transformation in question should be performed at a certain place or not. It is guaranteed that the decision is based on features of exactly the nodes involved in the transformation; and features used in smaller transformations probably fit into memory. The problem is that we would have to have such amount of data, that the training would take enormous time.

[26] The consequence of this decision is that I effectively lost about 6% of training data—which is not much.

- Abandoning this idea and trying to develop a small universal set of features which are relevant for making decisions about performance of every transformation—or, at least, of the most common ones. This seemed to be the only passable way for me.

When we look at the top twelve transformations, we can see that besides the node B, which a transformation is recorded along with, only its parent, some of its siblings on the right, and its children are involved in a transformation. Based on this observation, I decided to use five groups of nodes: the node in question, its parent, its children, its siblings on the left (they cannot occur in a transformation but they can bear valuable information) and its siblings on the right. After previous experiences, I did not attempt to select the set of features of these nodes nor their admissible combinations carefully; instead I used features from all layers, namely lemma, m-tag, s-tag, and functor of all these nodes, and left the choice of features on the template reduction. Beforehand I had to find out how to ensure the fixed number of features when I wanted to use features of a variable number of nodes. I decided to combine one feature of all nodes of any group into one. For example, if a node has two siblings on the right with functors *PAT* and *TWHEN*, functor representing group of its siblings on the right is *PAT+TWHEN*. This value is atomic for *fnTBL*. This way, I got 20 features.

With respect to this higher number I also wanted to use rule templates with relatively high number of features, so that the positive effect of combination of features could be indicated. I chose templates containing five features at maximum; however, the computation did not fit into memory even for a part of the training data under these conditions. I was forced to omit several features—I chose the lemma of children and siblings on the left and right, mainly for their sparseness. I also had to reduce the number of templates by eliminating those containing more that two features of a group. Templates created from the noted set of features according to the described constraints[27] were reduced with the only modification of the previous usage of this mechanism: since the number of resulting templates was low, all the templates whose instance occurred in the rule file were selected.

3.6.4 Experiments, results and statistics

Under the described conditions, the result of the structure assignment was $P_{\texttt{parent}} = 91.4\%$, $R_{\texttt{parent}} = 80.3\%$, $F_{\texttt{parent}} = 85.5\%$ and the result of the functor

[27] The way of obtaining these templates was not as short and direct as described—it took quite a long time to fit the computation into memory while constraining the templates as little as possible. It also showed me that the less constrained templates, the better results were obtained.

assignment $P_{\texttt{functor}} = 85.9\%$, $R_{\texttt{functor}} = 75.5\%$, $F_{\texttt{functor}} = 80.4\%$. The result in functor assignment is almost the same as after the previous phase, because functors were assigned just to new nodes in this phase.

On the left-hand side of Figure 3.7, there is the tectogrammatical annotation of the example sentence after this phase. Both the needed transformations consisting of movements of nodes *i* and *dvacet* are performed correctly.

I experimented with using two-letter m-tag, s-tag, and functor as the characteristics of nodes (see 3.6.2). The best ones are s-tags; however, the worst result was only slightly worse ($\Delta F_{\texttt{parent}} = -0.3\%$).

The final rule file was created using the setting `ORDER_BASED_ON_SIZE=1` and `threshold=1`; however, these settings have only insignificant impact on the result as compared to the default values. I found that using the `allPositiveRules` parameter proved to be damaging with all tested values.

In the training data, there are 2,658 unique transformations. Out of this number, only 141 got into the rule file and only 70 of them were then identified in the development-test data.

3.7 Phase 3: Creation of valency members

The aim of this phase is to create leaf nodes missing in the surface form of a sentence. These nodes typically correspond to obligatory valency slots. Moreover, the `val_frame.rf` attribute is assigned.

3.7.1 Design of the procedure

Although the `val_frame.rf` attribute contains the link to a description of the valency frame of a word and I could use values of this attribute for assignment of valency modifications, I decided not to do it. The reasons for this approach follow:

- Although the valency lexicon contains the valency frame appropriate to a word in its occurrence, this information is not complete. The missing part are rules transforming valency frames depending on certain properties of the word in question (e.g. passivization needs to be done when a verb is in passive voice) or of its valency members (e.g. when a member

is expressed with a numeral expression).[28] An implementation of the rules does exist, but it was created for the older data format and is not applicable any more; and even its conversion would be very elaborate. Moreover, the rules are tailored for the Czech language, which contradicts the claim of language independence of my analyzer.

- In other corpora, which my tool may process, it is not necessary to fill the attribute and a procedure based on its values then would not be of any use.
- Since an assignment of a valency frame is an assignment of an atomic value, the highest number of correctly assigned frames (this is what TBL mechanism would try to reach) does not necessarily mean the highest number of correctly assigned nodes (this is what the evaluation procedure judges). This might lower the accuracy of the tool.

Thus, instead of making a decision about valency members *en bloc* through the `val_frame.rf` attribute, I make decisions about each valency member separately.[29] The idea is simple: each valency slot corresponds to one feature which states whether (or, in case of free modifications, how many times) a member with the respective functor should occur along with an occurrence of a word. There is a question whether the feature should express the number of the respective members to be added, or their total number. The argument in favour of the latter approach is that rules are more independent of the members already expressed in the sentence, therefore consisting of less predicates and being more general. For this reason I chose the latter approach.

Since there is a lot of functors (67 of unique ones in the treebank) and only some of them can be assigned to obligatory valency members, I considered only those occurring at least twice in the training data along with nodes which I deemed to be valency members. For this task, I regarded every generated node with no children as a valency member. Mere 27 functors fulfill these criteria.

When the numbers of valency members are assigned, those missing in the trees are created and their functors are set. When there is an extra node in the tree, it is not deleted, though: this is not the task of this phase and, as implied by the analysis of errors, the node usually should exist—it is just misplaced.

However, things may be complicated by errors in the assignments of functors done in the first phase. If a wrong functor is assigned to a valency member, an

[28] The reason why this information is not expressed in the valency lexicon is that it is the same for all valency frames.

[29] Note that although I do not use the valency lexicon in its explicit representation, I do use it: it is implicitly present in the data used for training. For this reason I believe that the accuracy in both mentioned approaches would be approximately the same.

extra error can occur by creating a superfluous node with the correct functor in this phase. To avoid it, there is a need to design rule templates that are able to correct functors as well. This is a feasible place to do it, indeed: there are no intervening nodes between a word and its valency members here.

Of course, when merging PML-files with the output of the classifier, functors have to be corrected first, and, after that, nodes can be created.

As the last step of the procedure, common children are handled: when each member of a coordination has a generated node with a certain functor as its child, all these children are deleted and a node with that functor is created as a child of the coordination node.

3.7.2 Design of rule templates

Although valency is a property of a word, and thus lemma (*lemma*) should be definitely involved in the set of features, several of its characteristics can be inferred even from whole parts of speech, e.g. a verb usually has its actor. In order to be able to capture such relations, part of speech from the m-tag (*pos*) is another feature. Every rule template contains one of these two features.

Yet another property of a word affects its valency frame: voice by verbs (*voi*). Besides, negation of a verb is expressed at the t-layer by an extra node, rhematizer, which is child of the verb; this is why the information about a negation is also considered (*neg*). Thus, one or zero of these features, taken from the m-tag, are a part of rule templates. The described approach resulted in mere 162 rule templates.

Unlike in the case of adding the valency members, it seems more convenient to look at the corrections of functors from a different perspective: a correction will be recorded along with the node being altered, not along with its parent. The latter approach would lead to complicated scanning of features of children, whose number is variable and unlimited.

In order not to reinvent a wheel, I adopted same pieces of information used in the description of valency frames as features: the lemma of the word whose valency frame we are interested in (*plemma*) and the functor (*func*) and the morphemic realization of a valency member. Besides them, I used the voice of the parent (for the reason stated above) and the lemma of a valency member, hopefully useful for the identification of (not only) phrases. The features are combined in the following way: every rule template contains the lemma and either the functor, or the morphemic realization of a valency member. It can also contain its lemma and/or the voice of the parent. This gives another 8 templates and, because of their total number being low, there was no need to reduce templates.

For the assignment of `val_frame.rf`, the same set of templates was used with two extra templates: both contain the lemma of the word whose attribute is being assigned; one of them does not contain any other feature, the other contains the lemma of the valency member. I added these simple templates, since there was a need to assign the attribute, not only to correct it.

How a morphemic realization looks like remains to be explained. I have designed it to contain roughly the same information as that used in [Honetschläger, 2003]. In particular, it contains lemmas of all nouns, prepositions and conjunctions being auxiliary words, as well as part of speech and case (if the word exhibits it) of the autosemantic word.[30]

3.7.3 Experiments, results and statistics

After this phase, the result of the structure assignment was $P_{\texttt{parent}} = 90.2\%$, $R_{\texttt{parent}} = 87.9\%$, $F_{\texttt{parent}} = 89.0\%$ and the result of functor assignment was $P_{\texttt{functor}} = 86.5\%$, $R_{\texttt{functor}} = 84.3\%$, $F_{\texttt{functor}} = 85.4\%$. The result of the assignment of `val_frame.rf` is given in Section 3.8. The annotation of the example sentence after this phase is not stated; however, it differs only in the `t_lemma` attribute from the one after Phase 4, which is shown on the right-hand side of Figure 3.7 (speaking about attributes visualised in the figure).

Twelve of the most useful rules (without rules for `val_frame.rf`—they would be of no use for a reader) are shown in Table 3.3.

```
pos=V => ACT=1
pos=V neg=N => RHEM=1
pos=V voi=P => PAT=1
pos=V => RSTR=0
lemma=říci => ADDR=1
lemma=lze => BEN=1
plemma=jít func=PAT => func=ACT
lemma=říci => EFF=1
lemma=říkat => ADDR=1
lemma=jít => PAT=0
plemma=<root> func=RSTR lemma=mít => func=PRED
lemma=jednání => ACT=1
```

Table 3.3: The best rules in the third phase

[30] From this section it follows that I consider every word (except for coordination nodes) to be a valency member of its parent. One can truly object that this is not correct. Unfortunately, since I am not able to recognize whether a node exists because of valency or for another reason, I cannot choose but consider all words to be valency members and use the term "valency" in this broader sense.

As usual, I made several experiments. Their description follows.

I had the idea that presence or absence of a morphemic realization among found valency members could help distinguish which valency frame of a word is concerned. Thus, the addition of missing nodes and the assignment of functors of existing nodes could be more precise. For example, verb *jednat* (*transact/treat/proceed*) has three valency frames, one for each of its meanings:[31]

- *ACT*(1) *ADDR*(*s* (*with*)+7) *PAT*(*o* (*about*)+6) for the meaning of *vyjednávat* (*transact*),
- *ACT*(1) *PAT*(*s* (*with*)+7) *MANN*( ) for the meaning of *zacházet* (*treat*), and
- *ACT*(1) *MANN*[ ] for the meaning of *konat* (*proceed*).

When a node with the morphemic realization of *o*+6 is seen, the first frame is concerned and a node with morphemic realization of *s*+7 has to have *ADDR* as its functor, or, unless such a node exists, a node with functor *ADDR* has to be added. Unfortunately, since a node newly added to the t-layer has no morphemic realization, training data would differ in cases when a node with *s*+7 is present and when it is absent. This would cause the original rule "if *o*+6 is found, *ADDR* is *s*+7", interpreted in the way stated above, to crumble into two more specific and more complicated rules "if *o*+6 is found and there is no *s*+7, create an *ADDR*" and "if *o*+6 and *s*+7 exist, *ADDR* is *s*+7". Moreover, two other factors, which would presumably lead to low usability of suggested rule templates, deterred me from implementing them: after the examination of the data and the valency dictionary I find that there are probably too few morphemic realizations which can distinguish valency frames, and that nodes which can be identified by them often exist already. The latter factor can be illustrated using the verb *připravit* (*prepare/dispel*), which has two frames:

- *ACT*(1) *PAT*(4) for the meaning of *přichystat* (*prepare*), and
- *ACT*(1) *ADDR*(4) *PAT*(*o* (*of*)+4) for the meaning of *ukrást* (*dispel*).

The morphemic realization *o*+4 suggests that *ADDR* exists; however, it always exists at the a-layer already.

Inspired by the idea above, I tried to add rule templates conditioning the occurrence of a functor with another functor, not with a morphemic realization. I did not expect much of it, but the addition brought no extra work to me. It was not a surprise that these templates were unavailing.

[31] The morphemic realization of a valency member is written after its functor in parentheses if the member is obligatory, or in brackets if it is optional. A morphemic realization is expressed with a number denoting the case. In front of it, there is possibly a preposition delimited with a plus-sign. An empty realization means that the realization should be typical for the given functor.

When rules state that a node should have the same number of children as it has, but the functor of exactly one child differs, there are basically two possibilities: the child is misplaced and should be relocated and another node should be created instead of it; or, the child is just mistagged and its functor should be corrected. When I tried to correct the functor instead of creating a node in this situation, the result got slightly worse ($\Delta F_{\texttt{parent}} = -0.1\%$, $\Delta F_{\texttt{functor}} = -0.2\%$). Other experiments based on similar ideas led to even worse results.

The final rule file was created using the `ORDER_BASED_ON_SIZE=1` and `threshold=1` settings, although these settings have only an insignificant impact on the result as compared to the default values. The `allPositiveRules` parameter proved to be damaging with all its tested values.

3.8 Phase 4: Assignment of attributes

The aim of this phase is to perform the assignment of attributes `t_lemma`, `nodetype` and `gram/sempos`.

3.8.1 Design of the procedure

Assignment of the appointed attributes is a pure classification task, perfectly suitable for *fnTBL*.

There is a complication with the assignment of `t_lemma`, though: different pieces of information are important for this task depending on whether a node originates from the a-layer or not. In order to be able to design separate sets of templates for both cases, I had to divide the task into two classifications (*lem1* when a node originates from the a-layer, *lem0* otherwise). Of course, only one of the classifications is performed for each node, the other is set aside by initialization of the respective class to the "correct", dummy value. Then, the information about the origin of the node (*alay*) has to be a part of each template, so that the dummy value of the other class is not overwritten by useless rules. This fact is not stressed below any more.

Values of the `gram/sempos` attribute are compound; however, because of their relatively low number (19), I consider them atomic.

3.8.2 Design of rule templates

For nodes originating from the a-layer, there are two templates for assigning its `t_lemma`: the first considers only the previous value of this attribute (taken from lemma); the other considers lemmas of synsemantic words belonging to the node in question as well. The latter one is useful especially in case of verbs with reflexive pronouns.

For the assignment of `t_lemma` of nodes not originating from the a-layer, other pieces of information are important. The functor of the node in question (*func*) is included in each template. The templates also contain zero, one or both features of the parent of the node: its detailed part of speech (*pspos*) and information whether it is a finite verb, considering compound verb forms (*finv*).[32] The attribute is initialized based on functors.

For the assignment of `nodetype`, information about the origin of the node, its functor, its `t_lemma`, and the detailed part of speech of its parent are used and each template contains one or two of these features. The attribute is initialized based on the functor and the origin.

When assigning `gram/sempos`, the same features as in case of `nodetype` together with the detailed part of speech of the node (*spos*) proved useful. A template contains up to three of these features. The attribute is initialized based on the detailed part of speech of the node.

3.8.3 Experiments, results and statistics

Twelve of the most useful rules for each attribute being assigned are shown in Tables 3.4 to 3.6.

Table 3.7 contains the final results of my tool for tectogrammatical analysis running on PDT 2.0. The results are shown for all the attributes assigned, be they assigned both by means of training or by hand-coded procedures. Tests on d-test data as well as on e-test data were performed.[33]

The table also contains results obtained by the tool when it was trained and run on the same sets of data, which were automatically annotated at the m-layer and a-layer. The parser used for the annotation at the a-layer was the MST parser described in [McDonald et al., 2005] and it achieved accuracy of 84.2% on automatically annotated d-test data of PDT 2.0.[34] Errors my tool makes are analysed in Section 3.9.

[32] I do not analyze why features mentioned in this section as important are important—it usually results from the annotation manual.

[33] Needless to say that evaluation data were tested as late as the tool was in its final version.

[34] Note that in PDT 2.0, the sets of data at the a-layer are supersets of the data at the t-layer.

```
alay=0 func=ACT => lem0=#PersPron
alay=0 func=ACT finv=0 => lem0=#Gen
alay=1 lem1=, => lem1=#Comma
alay=0 func=RHEM => lem0=#Neg
alay=0 func=ADDR => lem0=#Gen
alay=0 func=ACT finv=0 pspos=f => lem0=#Cor
alay=0 func=PAT => lem0=#Gen
alay=1 lem1=on => lem1=#PersPron
alay=1 lem1=jeho => lem1=#PersPron
alay=0 func=ACT pspos=s => lem0=#Gen
alay=1 lem1=svůj => lem1=#PersPron
alay=1 lem1=: => lem1=#Colon
```

Table 3.4: The best rules for `t_lemma` (fourth phase)

```
alay=1 => nodetype=complex
alay=0 => nodetype=qcomplex
func=CONJ => nodetype=coap
func=RHEM => nodetype=atom
func=ACT pspos=p => nodetype=complex
func=ACT pspos=B => nodetype=complex
func=PREC => nodetype=atom
func=APPS => nodetype=coap
func=FPHR => nodetype=fphr
func=CM => nodetype=atom
lem1=ale => nodetype=coap
func=ATT => nodetype=atom
```

Table 3.5: The best rules for `nodetype` (fourth phase)

On the right side of Figure 3.7, there is the final tectogrammatical annotation of the example sentence. The nodes are added correctly except for the extra node with *PAT* functor belonging to *obava*. The functors of the two nodes mistagged in the Phase 1 remained unrepaired. The correct tectogrammatical annotation is shown in Figure 3.8.

3.9 Analysis of errors

In Table 3.7 we can see that when the input data are annotated automatically instead of manually at lower layers, the falls of the F-measures corresponding to particular attributes differ in a great deal. The explanation is that it is the

```
spos=N => sempos=n.denot
spos=A => sempos=adj.denot
alay=1 func=PRED => sempos=v
spos=B => sempos=v
spos== => sempos=adj.quant.def
spos=p => sempos=v
spos=f => sempos=v
spos=4 => sempos=n.pron.indef
spos=g => sempos=adj.denot
spos=D => sempos=adj.pron.def.demon
alay=0 func=ACT pspos=p => sempos=n.pron.def.pers
spos=S => sempos=n.pron.def.pers
```

Table 3.6: The best rules for `gram/sempos` (fourth phase)

analytical structure what is assigned most incorrectly in the input data—and the fall of performance of assignment of an attribute roughly corresponds to dependency of values of the attribute on the analytical structure. The biggest fall is seen in case of the tectogrammatical structure, which originates from the analytical one. Similarly with functors: they express relation between nodes and thus they need to have dependencies assigned correctly to be able to describe the relations correctly. Values of `is_member` are dependent on the structure to a certain extent as well, and the fall of the respective performance is also remarkable.

The statistics proposed in this section are gathered on the d-test data manually annotated at m-layer and t-layer—the golden annotation contains 71,306 nodes, my test annotation contains 69,445 nodes and 68,162 of them are mutually aligned.

Unless stated otherwise, for each attribute being assigned by my tool by means of training, two tables describing errors in its assignment are presented. The first one gives a sorted list of the most frequent errors: the assigned incorrect value is followed with an arrow and the correct value; number of errors of this kind is given as well. A special value "none" means that no value of the attribute is assigned to a node—it also can mean that the node does not exist. For example, the first record in Table 3.8 states that 920 nodes with no value of the functor should have their functors set to value *ACT*—and since my tool sets functor of each node, it means that 920 nodes should have been created (or should not have been deleted) and their functors should have been set to *ACT*. Similarly, the fifth record states that 352 nodes having *ACT* as their functor should have been deleted (or should not have been created).

attribute	manual		automatic	
	d-test	e-test	d-test	e-test
structure	89.0%	89.3%	77.1%	76.4%
functor	85.4%	85.5%	77.8%	77.4%
`val_frame.rf`	92.3%	92.3%	91.0%	90.9%
`t_lemma`	93.5%	93.5%	91.0%	90.9%
`nodetype`	94.4%	94.5%	92.7%	92.6%
`gram/sempos`	93.5%	93.8%	91.6%	91.5%
`a/lex.rf`	96.5%	96.5%	95.3%	95.1%
`a/aux.rf`	94.2%	94.3%	90.4%	90.3%
`is_member`	94.1%	94.3%	89.8%	89.5%
`is_generated`	96.5%	96.6%	95.3%	95.2%
`deepord`	67.6%	68.0%	66.5%	66.7%

Table 3.7: The final F-measure of tectogrammatical annotation

#	auto → correct	#	auto → correct
920	none → *ACT*	204	*APP* → *ACT*
520	none → *PAT*	200	*APP* → *PAT*
373	*ACT* → *PAT*	166	*DENOM* → *ACT*
358	none → *PRED*	149	none → *ADDR*
352	*ACT* → none	134	*PAT* → *APP*
340	*PAT* → none	130	*RSTR* → *PAT*
330	*PAT* → *ACT*	129	*PAT* → *EFF*
265	none → *RHEM*	116	*RHEM* → *CM*

Table 3.8: The most frequent errors in the assignment of functors and their counts

The second type of tables expresses the probability that a value of the respective attribute of a node in the test annotation is correct. For example, in Table 3.9 we can see that 89.4% of nodes in the test annotation having their functors set to *ACT*, have their functors set correctly—thus, 10.6% of such nodes should have their functors other than *ACT* or they should not exist at all.

As stated above, Tables 3.8 and 3.9 capture errors in assignment of functors. From Table 3.8 we can see that most of mistakes is caused by superfluous or missing nodes. As for mistagging, mismatch between *ACT* and *PAT* is the most frequent, although these values are relatively reliable (see Table 3.9). This is caused by high frequency of their occurrence. Mismatches of one of these values and *APP* is also frequent. The most frequent errors captured in Table 3.8 cover 37% of occurences of errors.

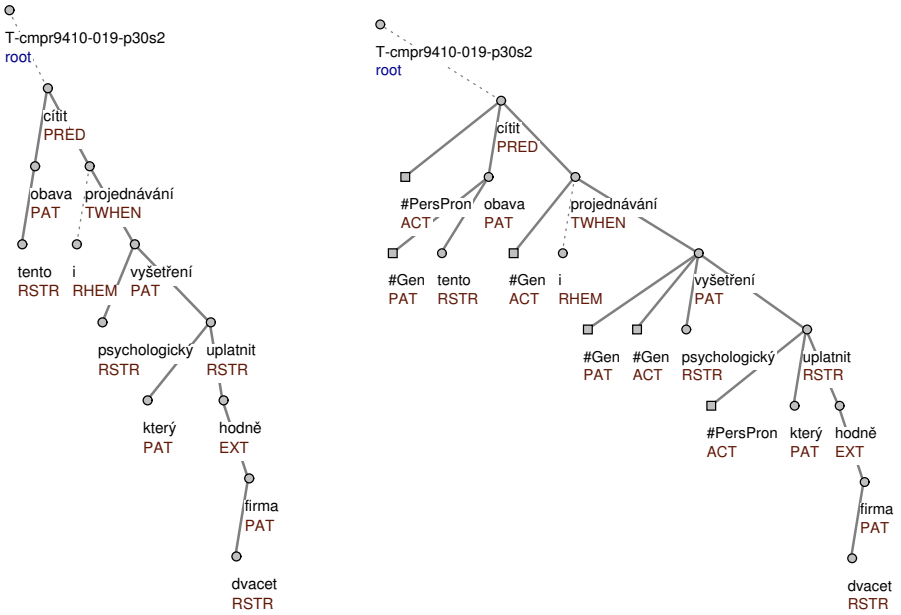


Figure 3.7: The sentence from Figure 3.3 after Phase 2 (left) and after Phase 4 (right)

Table 3.10 captures errors in assignment of valency frames. *w243* in an identifier means, that the frame belongs to the word *být* (*to be*), *w1855* denotes *mít* (*to have*), *w7646* denotes *vlastní* (*own*), *w5886* denotes *řízení* (*trial/proceeding*), *w327* denotes *činit* (*make*), *w9501* denotes *získat* (*get*), and finally *w7548* denotes *vědět* (*know*). The table captures mere 12% of errors—over 1,600 of errors occur only once. The second type of table for this attribute is not presented because of its enormous size.

In Table 3.11 we can see errors in assignment of the `t_lemma` attribute. When we leave errors caused by existence or nonexistence of nodes aside, the most important source of errors is mismatch between *#PersPron* (personal pronoun) and *#Gen* (general participant). The errors listed in the table cover 54% of errors. For the same reason as in case of valency frames, the second type of table is not presented for the `t_lemma` attribute.

Tables 3.12 and 3.13 summarize errors made during assignment of the attribute `nodetype`. Besides errors caused by errors in structure, the most important mismatch is *complex* versus *qcomplex*. Table 3.12 captures 89% of errors in this attribute.

In Tables 3.14 and 3.15 we can see that assignment of the attribute `sempos` suffers, besides usual problems with the structure, mainly from mismatch be-

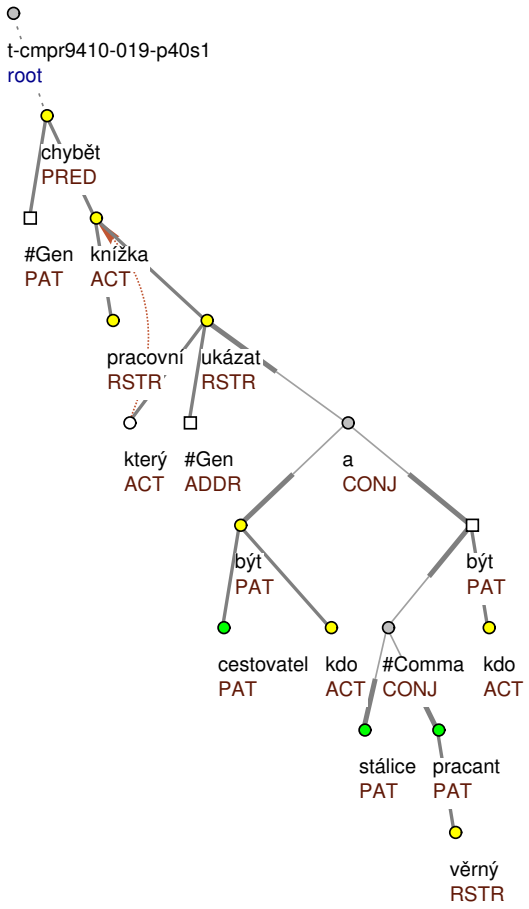


Figure 3.8: The correct t-layer annotation of the sentence from Figures 3.3 and 3.7

tween semantic nouns (*n*) and adjectives (*adj*); other characteristics (e.g. definiteness) are usually assigned well. Table 3.14 covers 83% of errors. Note that the value of this attribute need not to be filled for all nodes—this is also why probability of the case that the corresponding value "none" is assigned well is greater than zero.

Tables 3.16 and 3.17 describing errors in assignment of structure are constructed the same way as all those above; however, due to the nature of this information, the description of nodes and of errors in assignment needs a different approach. I decided to describe a node as a combination of its topological characteristic and information about its origin. From the point of view of topology, I distinguish only two possibilities: whether a node is an inner node (*inner*), or whether it is a leaf (*leaf*). As for its origin, it is expressed in

VOCAT	100.0	*CPHR*	86.4	*ATT*	78.2	*ID*	67.9
REAS	100.0	*DPHR*	85.6	*EFF*	77.6	*DENOM*	67.1
CSQ	100.0	*LOC*	84.3	*TSIN*	76.8	*CONTRD*	66.7
CONFR	100.0	*DIR1*	84.1	*CNCS*	76.3	*SUBS*	65.4
RESTR	98.4	*COMPL*	84.0	*DIR3*	75.5	*DIR2*	63.6
PRED	97.6	*APPS*	83.6	*PAR*	74.9	*TFHL*	63.3
CRIT	95.9	*PAT*	83.5	*ACMP*	74.6	*ADVS*	61.3
RSTR	93.1	*DISJ*	82.8	*PARTL*	72.7	*RESL*	61.1
INTF	90.9	*CAUS*	82.4	*INTT*	72.7	*AIM*	60.3
MAT	90.3	*TWHEN*	82.0	*BEN*	72.6	*AUTH*	58.5
TTILL	89.7	*PREC*	81.5	*MANN*	72.5	*DIFF*	58.1
ACT	89.4	*ADDR*	81.4	*CPR*	72.5	*ORIG*	57.8
RHEM	89.0	*EXT*	80.3	*COND*	71.6	*TFRWH*	53.3
THO	88.4	*CONTRA*	80.0	*THL*	71.3	*TOWH*	52.6
MOD	88.4	*CM*	79.1	*REG*	71.1	*GRAD*	45.5
CONJ	88.1	*FPHR*	78.6	*OPER*	69.7		
TPAR	86.7	*APP*	78.6	*MEANS*	68.8		

Table 3.9: Reliability (in %) of the assignment of functors

attributes `is_generated` and `a/lex.rf` and three their following combinations are possible.

- `is_generated` is 0 (and `a/lex.rf` is then always filled)—this means that the node originates from the a-layer (denoted as *orig*),
- `is_generated` is set to 1 and `a/lex.rf` is not filled—the node is brand new at t-layer (denoted as *new*), or
- `is_generated` is set to 1 and `a/lex.rf` is filled—the node is new, but it is linked with the a-layer; shortly, it is a copy of a node at the t-layer (denoted as *copy*).

In Table 3.16, I decided not to capture errors in assignment as a comparison of characteristics of a node when it is placed incorrectly on one hand and correctly on the other hand (like in the previous tables of this type), but as the length of the path from the incorrect to the correct position of the node in question. The reason for this is that, unlike in the cases above, the characteristics of a node do not identify its position, and, moreover, I consider the latter approach useful enough: an annotator correcting an annotation made by my tool probably likes to know whether he or she has to search for the correct parent of a node in its neighbourhood, or whether the move of the node will be possibly longer. For a deeper insight into errors I split the path from incorrect

#	auto → correct	#	auto → correct
126	*v-w243f1* → *v-w243f2*	19	none → *v-w243f2*
86	none → *v-w243f1*	19	none → *v-w327f1*
36	*v-w243f1* → *v-w243f3*	16	*v-w1855f1* → *v-w1855f3*
29	*v-w1855f1* → *v-w1855f2*	14	*v-w9501f1* → *v-w9501f2*
29	none → *v-w7646f1*	13	*v-w7548f1* → *v-w7548f2*
28	*v-w243f2* → *v-w243f1*	12	*v-w5886f1* → *v-w5886f3*

Table 3.10: The most frequent errors in the assignment of the `val_frame.rf` attribute and their counts

#	auto → correct	#	auto → correct
715	none → *#Gen*	131	*#Gen* → *#PersPron*
527	*#Gen* → none	119	none → *#Forn*
297	none → *#PersPron*	100	*#Forn* → none
268	none → *#Neg*	87	none → *být*
266	*#PersPron* → none	72	*#Cor* → *#Gen*
262	*#PersPron* → *#Gen*	68	none → *#Comma*
233	none → *#EmpVerb*	67	*#Oblfm* → none
218	none → *#QCor*	65	*#Gen* → *#Rcp*
131	none → *#Oblfm*	57	none → *#Cor*

Table 3.11: The most frequent errors in the assignment of the `t_lemma` attribute and their counts

to correct position into three parts: how far a node should be moved up to the root; how far it should be then moved down; and how far it should be moved down the nodes non-existing in the (test) tree. The triplet of these numbers in the stated order is given on the right-hand side of arrows. For example, the triplet *3,1,1* means that to correct the position of the respective node, it should be moved three times up and then twice down, but on the way down one more node should have existed. From Table 3.16 we can see that the most frequent errors in structure are very local. The table covers 88% of errors.

3.10 Assignment of functors only

Inspired by [Sgall et al., 2002], I also tried to assign functors once the tectogrammatical structure is correct. Since I can profit from the existing structure now, the choice of features differs from that in 3.5. The first substantial difference is that the parent considered is the effective one. As before, I used the lemma and s-tag of the node in question and of its parent, but only several po-

#	auto → correct	#	auto → correct
1490	none → *qcomplex*	276	none → *atom*
1077	none → *complex*	226	*complex* → *fphr*
694	*qcomplex* → none	184	*complex* → *atom*
422	*qcomplex* → *complex*	167	*atom* → *complex*
381	*complex* → none	163	none → *list*
293	*complex* → *qcomplex*	118	none → *coap*

Table 3.12: The most frequent errors in the assignment of the `nodetype` attribute and their counts

complex	98.0	*fphr*	78.9
coap	93.9	*qcomplex*	73.3
atom	93.4	*list*	28.7
dphr	86.5		

Table 3.13: Reliability (in %) of the assignment of the `nodetype` attribute

sitions of their m-tags: the part of speech, the detailed part of speech, the case, and the voice (in order to reduce the possible number of templates). Moreover, I added number of children of both nodes (distinguishing only values 0, 1 and more for the child and 1, 2 and more for the parent) and the morphemic realization of the child—without its case, since this is a separate feature already. As I initialized the functors (to the most probable value corresponding to the pair of s-tags of both nodes, which gives baseline accuracy of 61.8%), I wanted the rules to make a decision based even on their values. This is why the functor of a node in question was incorporated into the features as well. Unlike the authors of the cited article, I did not use ontological attributes, since I did not consider them to be worth the effort because of their rather modest contribution to the accuracy.

The rule templates are designed in such a way that at most three of these features can occur in a rule and at least one of them has to refer to the node whose functor is being assigned. Since the number of templates obtained this way is high, template reduction is used, retaining the templates which have at least two instances in the rules obtained using smaller data.

Similarly to the authors of [Sgall et al., 2002], I excluded nodes having no a-layer counterpart. My resulting accuracy is 90.0%, which means 43% error reduction as compared to their one. In order to allow the comparison of my results to their ones, I tried to perform training on 24,734 nodes randomly chosen, which was 9/10 of the amount that the authors had at their disposal

#	auto → correct	#	auto → correct
645	*none* → *n.pron.def.pers*	173	*adj.quant.def* → *n.quant.def*
560	*n.pron.def.pers* → none	117	*n.quant.def* → *adj.quant.def*
373	none → *n.denot*	105	none → *adv.denot.ngrad.nneg*
357	none → *v*	94	none → *n.pron.def.demon*
301	*n.denot* → *n.denot.neg*	94	*adj.denot* → none
267	*n.denot* → none	62	*adj.denot* → *n.denot*

Table 3.14: The most frequent errors in the assignment of the `sempos` attribute and their counts

adv.denot.grad.nneg	100.0	*adv.denot.ngrad.neg*	95.5
n.denot.neg	99.8	*adj.pron.indef*	93.9
adv.pron.indef	99.7	*adv.denot.ngrad.nneg*	93.8
v	99.3	*adv.denot.grad.neg*	91.4
adj.quant.indef	98.7	*adj.quant.def*	91.1
adj.quant.grad	98.5	none	88.5
adj.denot	98.4	*n.pron.def.demon*	84.5
n.denot	97.5	*n.pron.def.pers*	83.2
n.pron.indef	97.2	*n.quant.def*	82.0
adj.pron.def.demon	96.2	*adv.pron.def*	81.6

Table 3.15: Reliability (in %) of the assignment of the `sempos` attribute

for training and testing purposes. In that case, the accuracy was 82.7% as compared to their 82.3%.

The final rules were generated using `ORDER_BASED_ON_SIZE=1` and `threshold 1` settings.

When the templates were extended so that they could contain up to four features, the accuracy remained the same. This suggests that the method, as it is designed, may reached its limit.

3.10.1 Effects of template reduction

In Section 3.5 I introduced the **template reduction**, a technique lowering memory requirements when training, and used it, beside others, for the assignment of functors, as described above. In this case, it is not necessary to use it, since the training with the full template set can fit into memory. I would like to employ this fact and compare several aspects of training using the full template set and the reduced one.

#	auto → correct	#	auto → correct
1751	none → *leaf, new*	451	none → *inner, new*
1036	*leaf, new* → none	413	*leaf, orig* → *0,0,1*
869	*leaf, orig* → *0,1,0*	292	*leaf, orig* → *1,1,0*
754	*inner, orig* → *0,0,1*	264	none → *inner, orig*
727	*inner, orig* → *1,0,0*	187	*inner, orig* → *1,1,0*
664	*inner, orig* → *0,1,0*	139	*inner, orig* → *0,0,2*
486	none → *inner, copy*	136	none → *leaf, orig*
463	*leaf, orig* → *1,0,0*	122	*inner, orig* → none

Table 3.16: The most frequent errors in the assignment of structure and their counts

leaf, copy	93.3	*inner, new*	84.1
leaf, orig	91.9	*leaf, new*	79.9
inner, orig	91.0	*inner, copy*	69.4

Table 3.17: Reliability (in %) of the assignment of structure

The comparison is shown in Table 3.18. In the case of template reduction, information about both needed computations is given where it makes sense. The reduced template set was obtained with one eighth of the training data; all the computations were performed with the settings described above. Both resulting rule files were used for labelling the whole d-test data.

Template set	full	reduced
number of templates	633	122
used memory (MB)	2452	488 and 869
time of training (minutes)	1065	352 (53+299)
number of rules	9691	9328
number of used templates	325	121
resulting accuracy	89.8%	90.0%

Table 3.18: Effects of template reduction

In this table, we can see that not only memory requirements, but, not surprisingly, also the time needed for training decreased to just about one third. Unlike the training using the reduced set, the training using the full set of templates did not make instances of all templates by far which it had to its disposal. Even so, for the obvious reason, the number of templates with at least one instance was much higher in the case of the full set. This is also the explanation of the number of generated rules in the former case be-

ing slightly higher—and this is also the explanation of the fact, which may be surprising at the first glance, that with the reduced set, the accuracy is higher.[35] More elaborately: lower number of templates can cause both lower and higher accuracy—the latter one because of preventing the overtraining, which is probably what happened in our case. We rejected using very infrequent templates—and they were rare because they were not able to learn linguistically motivated information, only noise.

3.10.2 Dependency between the amount of training data and accuracy

The task of functor assignment is also suitable for detecting the relation of the amount of the training data and resulting accuracy—it has only one phase, there is no need to use the template reduction nor there is, as opposed to e.g. analytical parsing, any important post-processing phase.

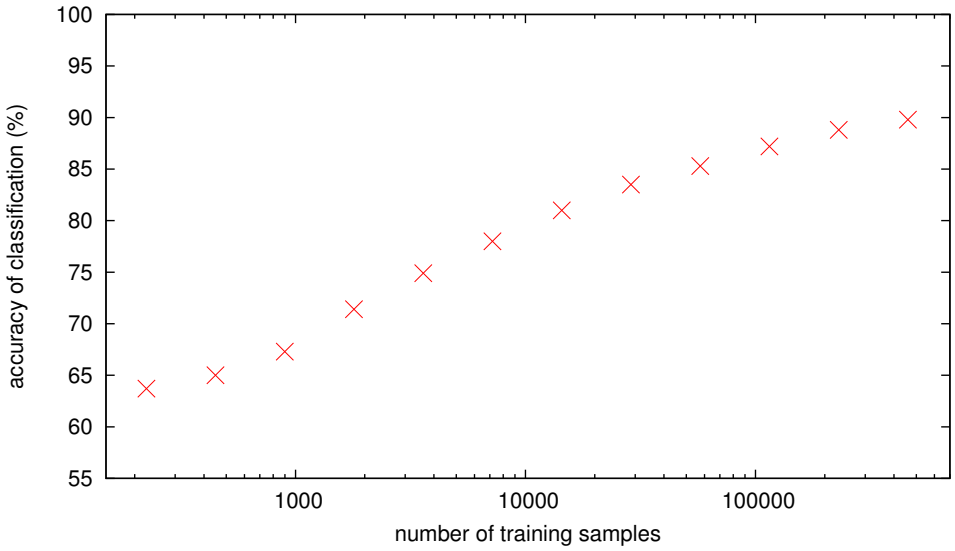


Figure 3.9: Dependency between the number of training samples and accuracy

Figure 3.9 shows this relation. The results are obtained without the template reduction and with the setting given above. Beside the whole training data, the computation is performed on their half, quarter, eighth etc. Since I sup-

[35] I would like to stress that even in other tasks, whenever the results using a full template set on one hand and a reduced template set on the other hand were comparable, the reduced set performed, with reasonable settings, at worst as well as the full one.

posed an approximately logarithmic dependency, the x-axis has the logarithmic scale.

In this figure, we can see that even for small data the accuracy does not fall below 60%. On the other hand, if we double the size of our current training data, we can expect at most 1% improvement.

Chapter 4

Conclusion

'There's glory for you!' 'I don't know what you mean by "glory",' Alice said.
Humpty Dumpty smiled contemptuously. 'Of course you don't—till I tell you.
I meant "there's a nice knock-down argument for you!" '
'But "glory" doesn't mean "a nice knock-down argument",' Alice objected.
'When I use a word,' Humpty Dumpty said, in rather a scornful tone,
'it means just what I choose it to mean—neither more nor less.'
'The question is,' said Alice, 'whether you can make words mean so many different things.'
'The question is,' said Humpty Dumpty, 'which is to be master—that's all.'

[Carroll, 2003]

4.1 Conclusions (analytical analysis)

In this thesis, two parsers and a tool for assigning syntactic tags are proposed.

The parsing Method 1 has serious drawbacks, since rules produced by it can hardly be extended to operate with another type of information than they do. Moreover, the method is hardly configurable—e.g. one cannot make a trade-off between accuracy on one hand and speed or memory requirements on the other hand. Despite its drawbacks, many ideas the method is based on are used in a more successful and very flexible Method 2.

The performance of both parsers is about 10% below the state-of-the-art parsers, but both methods the parsers are based on are novel—the first one is completely new; the second approach, rule-based classification, has never been used for dependency parsing to my knowledge. Improvements of the more promising of the parsers are suggested as well; some of them could be used in other parsing methods as well.

The parser based on Method 2 is designed to be as independent of the processed language as possible: when one wants to adapt the parser for processing other language than Czech, only the following adjustments apply in general: the choice of relevant positions of m-tags needs to be redone (supposing that the m-tag is positional), the choice of rule templates may be altered, and, for languages allowing more non-projectivities than Czech does, the usage of the respective post-processing procedure should be reconsidered—and, of course, the parser has to be retrained.[1]

However small the improvement of the assignment of syntactic tags over the previous tool seems to be, for those very high accuracies it corresponds to an considerable 15% error reduction.

4.2 Conclusions (tectogrammatical analysis)

The tool I have developed is currently the only one that can perform tectogrammatical annotation in such a breadth—it can assign the tectogrammatical structure and many of the most important attributes describing the tectogrammatical layer. Although other (specialized) tools probably perform some particular subtasks better than my tool does, for Czech it outperforms the combination of existing tools annotating the tectogrammatical structure and deep functors (both of which can be considered the core of the tectogrammatical layer) by 29% and 47%, respectively, measured as a relative error reduction of F-measure.

Moreover, the tool is designed to be as much independent of Czech as possible. The issues of its language independence are mentioned and spots where the tool should be adjusted for processing another language are identified. However, due to the lack of needed annotated corpora of other languages, the tool could not be engaged on a new language and neither its language independence nor its performance on another language could be tested.

I also developed a technique for decreasing the time and memory requirements of the training phase of the tool, when they are caused by a high count of potentially useless rule templates.

A utility for evaluation the tectogrammatical annotation is proposed and implemented as well.

The tool is presently used at our Institute for preannotation of data whenever needed for their subsequent manual annotation.

[1] This is exactly the way how I developed the first version of a fully functional analytical parser of a PDT-like annotated Latin corpus in no more than 10 minutes.

Index

Bibliography

Ondřej Bojar. Towards Automatic Extraction of Verb Frames. *Prague Bulletin of Mathematical Linguistics*, 79–80:101–120, 2003. ISSN 0032-6585. URL `http://ufal.mff.cuni.cz/pbml/79-80/bojar.pdf`.

Eric Brill. A Simple Rule-Based Part-of-Speech Tagger. In *Proceedings of ANLP-92, 3rd Conference on Applied Natural Language Processing*, pages 152–155, Trento, Italy, 1992. URL `http://www.cs.mu.oz.au/acl/A/A92/A92-1021.pdf`.

Eric Brill. Transformation-Based Error-Driven Parsing. In *Proceedings Third International Workshop on Parsing Technologies*, Tilburg/Durbuy, 1993. URL `http://www.cs.jhu.edu/~brill/Parsing_Tech_93.ps`.

Alena Böhmová. Automatic Procedures in Tectogrammatical Tagging. *Prague Bulletin of Mathematical Linguistics*, 76:23–34, 2001. ISSN 0032-6585. URL `http://ufal.mff.cuni.cz/pdt2.0/publications/BohmovaPBML2001.pdf`.

Lewis Carroll. *Alice's Adventures in Wonderland and Through the Looking-Glass*. Penguin Classics, 2003. ISBN 9780141439761.

Eugene Charniak. A Maximum-Entropy-Inspired Parser. In *Proceedings of NAACL*, Seattle, Washington, 2000. URL `http://www.cs.brown.edu/~ec/papers/shortMeP.ps.gz`.

Michael Collins, Jan Hajič, Lance Ramshaw, and Christoph Tillmann. A Statistical Parser for Czech. In *Proceedings of the 37th Annual Meeting of the ACL*, College Park, Maryland, 1999. URL `http://people.csail.mit.edu/mcollins/papers/acl99.ps`.

Radu Florian, John C. Henderson, and Grace Ngai. Coaxing Confidences from an Old Friend: Probabilistic Classifications from Transformation Rule Lists. In *Proceedings of EMNLP 2000*, pages 26–34, Hong Kong, 2000. URL `http://nlp.cs.jhu.edu/~rflorian/papers/emnlp00.ps`.

Radu Florian and Grace Ngai. *Fast Transformation-Based Learning Toolkit*. Baltimore, MD, 2001a. URL http://nlp.cs.jhu.edu/~rflorian/fntbl/tbl-toolkit/tbl-toolkit.html.

Radu Florian and Grace Ngai. Multidimensional transformation-based learning. In *Proceedings of CoNLL 2001*, pages 1–8, Toulouse, France, 2001b. URL http://nlp.cs.jhu.edu/~rflorian/papers/conll01.ps.

Kadri Hacioglu. Semantic Role Labeling Using Dependency Trees. In *Proceedings of Coling 2004*, pages 1273–1276, Geneva, Switzerland, Aug 23–Aug 27 2004. URL http://sds.colorado.edu/SERF-II/papers/hacioglu_coling2004.pdf.

Jan Hajič, Yuan Ding, Jason Eisner, Martin Čmejrek, Terry Koo, Kristen Parton, Gerald Penn, Drago Radev, and Owen Rambow. Natural Language Generation in the Context of Machine Translation, Workshop '02 Final Report. Technical report, CLSP Technical Reports, Baltimore, MD, 2003. URL http://www.clsp.jhu.edu/ws2002/groups/mt/genmt-final.ppt.

Jan Hajič, Eva Hajičová, Jaroslava Hlaváčová, Václav Klimeš, Jiří Mírovský, Petr Pajas, Jan Štěpánek, Barbora Vidová Hladká, and Zdeněk Žabokrtský. Prague Dependency Treebank 2.0. CDROM, 2006. URL http://ufal.mff.cuni.cz/pdt2.0/. In press.

Jan Hajič, Eva Hajičová, Petr Pajas, Jarmila Panevová, Petr Sgall, and Barbora Vidová Hladká. Prague Dependency Treebank 1.0. CDROM, 2001. URL http://ufal.mff.cuni.cz/pdt/. CAT: LDC2001T10.

Jan Hajič and Václav Honetschläger. Annotation Lexicons: Using the Valency Lexicon for Tectogrammatical Annotation. *Prague Bulletin of Mathematical Linguistics*, 79–80:61–86, 2003. ISSN 0032-6585. URL http://ufal.mff.cuni.cz/pdt2.0/publications/HajicHonetschlagerPBML2003.pdf.

Eva Hajičová, Zdeněk Kirschner, and Petr Sgall. A Manual for Analytic Layer Annotation of the Prague Dependency Treebank (English translation). Technical report, ÚFAL, MFF UK, Prague, Czech Republic, 1999. URL http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/en/a-layer/pdf/a-man-en.pdf.

Keith Hall and Václav Novák. Corrective Modeling for Non-Projective Dependency Parsing. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, Vancouver, British Columbia, 2005. Association for Computational Linguistics. URL http://ufal.mff.cuni.cz/publications/year2005/IWPT05-Hall.ps.

Tomáš Holan. Genetické učení závislostních analyzátorů [Genetic-based Learning of Dependency Parsers]. In Peter Vojtáš, editor, *Sborník semináře [Proceedings of] ITAT 2005*, pages 47–54, Košice, Slovakia, 2005. UPJŠ. In Czech.

Tomáš Holan and Zdeněk Žabokrtský. Combining Czech Dependency Parsers. In Petr Sojka, Ivan Kopeček, and Karel Pala, editors, *Proceedings of the 9th International Conference on Text, Speech and Dialogue*, Berlin Heidelberg New York, 2006. Springer-Verlag. Accepted.

Václav Honetschläger. Using a Czech Valency Lexicon for Annotation Support. In V. Matoušek and P. Mautner, editors, *Proceedings of the 6th International Conference on Text, Speech and Dialogue*, pages 120–125, Berlin Heidelberg New York, 2003. Springer-Verlag. ISBN 3-540-20024-X. URL http://ufal.mff.cuni.cz/publications/year2003/honet_tsd03final.ps.

Ivona Kučerová and Zdeněk Žabokrtský. Transforming Penn Treebank Phrase Trees into (Praguian) Tectogrammatical Dependency Trees. *Prague Bulletin of Mathematical Linguistics*, 78:77–94, 2002. ISSN 0032-6585. URL http://ufal.mff.cuni.cz/pbml/78/zabok-kucer.pdf.

Lucie Kučová and Zdeněk Žabokrtský. Anaphora in Czech: Large Data and Experiments with Automatic Anaphora. In Václav Matoušek, Pavel Mautner, and Tomáš Pavelka, editors, *LNCS/Lecture Notes in Artificial Intelligence/Proceedings of Text, Speech and Dialogue*, volume 3658, pages 93–98, Karlovy Vary, Czech Rep., Sept. 12-16, 2005. Springer Verlag Heidelberg. ISBN 3-540-28789-2. URL http://ufal.mff.cuni.cz/publications/year2005/tsd2005-coref.pdf.

Markéta Lopatková, Ondřej Bojar, Jiří Semecký, Václava Benešová, and Zdeněk Žabokrtský. Valency Lexicon of Czech Verbs VALLEX: Recent Experiments with Frame Disambiguation. In Václav Matoušek, Pavel Mautner, and Tomáš Pavelka, editors, *LNCS/Lecture Notes in Artificial Intelligence/Proceedings of Text, Speech and Dialogue*, volume 3658, pages 99–106, Karlovy Vary, Czech Rep., Sept. 12-16, 2005. Springer Verlag Heidelberg. ISBN 3-540-28789-2. URL http://ufal.mff.cuni.cz/publications/year2005/05-vallex-tsd.pdf.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of the Human Language Technology / Empirical Methods in Natural Language Processing conference (HLT-EMNLP)*, Vancouver, British Columbia, 2005. Association for Computational Linguistics. URL http://www.seas.upenn.edu/~ryantm/papers/nonprojectiveHLT-EMNLP2005.pdf.

Marie Mikulová, Alevtina Bémová, Jan Hajič, Eva Hajičová, Jiří Havelka, Veronika Kolářová-Řezníčková, Lucie Kučová, Markéta Lopatková, Petr Pajas, Jarmila Panevová, Magda Razímová, Petr Sgall, Jan Štěpánek, Zdeňka Urešová, Kateřina Veselá, and Zdeněk Žabokrtský. Anotace Pražského závislostního korpusu na tektogramatické rovině: pokyny pro anotátory [A Manual for Tectogrammatical Layer Annotation of the Prague Dependency Treebank]. Technical report, ÚFAL, MFF UK, Prague, Czech Republic, 2005. URL `http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/cz/t-layer/pdf/t-man-cz.pdf`. In Czech.

Grace Ngai and Radu Florian. Transformation-Based Learning in the Fast Lane. In *Proceedings of NAACL 2001*, pages 40–47, Pittsburgh, PA, 2001. URL `http://nlp.cs.jhu.edu/~rflorian/papers/naacl01.ps`.

Petr Pajas and Jan Štěpánek. A Generic XML-Based Format for Structured Linguistic Annotation and Its Application to Prague Dependency Treebank 2.0. Technical Report 29, ÚFAL MFF UK, Prague, Czech Republic, 2005. URL `http://ufal.mff.cuni.cz/pdt2.0/publications/PajasStepanekTR2005.pdf`.

Oana Postolache. Learning Information Structure in The Prague Treebank. In *Proceedings of the ACL 2005 Student Session*, pages 115–120, Ann Arbor, USA, 2005. URL `http://www.coli.uni-saarland.de/~oana/publications/postolache-05.pdf`.

Magda Razímová and Zdeněk Žabokrtský. Morphological Meanings in the Prague Dependency Treebank 2.0. In Václav Matoušek, Pavel Mautner, and Tomáš Pavelka, editors, *LNCS/Lecture Notes in Artificial Intelligence/Proceedings of Text, Speech and Dialogue*, volume 3658, pages 148–155, Karlovy Vary, Czech Rep., Sept. 12-16, 2005. Springer Verlag. ISBN 3-540-28789-2. URL `http://ufal.mff.cuni.cz/publications/year2005/tsd2005-grammatemes.pdf`.

Kiril Ribarov. *Automatic Building of a Dependency Tree—The Rule-Based Approach and Beyond*. PhD thesis, ÚFAL MFF UK, Prague, Czech Republic, 2004.

Giorgio Satta and Eric Brill. Efficient transformation-based parsing. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 255–262, San Francisco, 1996. Morgan Kaufmann Publishers. URL `http://research.microsoft.com/~brill/Pubs/Eff_Pars.ps`.

Jiří Semecký. On Automatic Assignment of Verb Valency Frames in Czech. In *Proceedings of LREC 2006*, 2006. In print.

Petr Sgall, Eva Hajičová, and Jarmila Panevová. *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. Reidel Publishing Company and Academia, Dordrecht and Prague, 1986.

Petr Sgall, Zdeněk Žabokrtský, and Sašo Džeroski. A Machine Learning Approach to Automatic Functor Assignment in the Prague Dependency Treebank. In C. Paz Suárez Araujo R. M. Rodríguez, editor, *Proceedings of the 3rd International Conference on Language Resources and Evaluation*, volume 5, pages 1513–1520. European Language Resources Association, 2002. URL http://ufal.mff.cuni.cz/pdt2.0/publications/ZabokrtskySgallDzeroski2002.pdf.

Dan Zeman, Jiří Hana, Hana Hanová, Jan Hajič, Barbora Hladká, and Emil Jeřábek. A Manual for Morphological Annotation, 2nd edition. Technical Report 27, ÚFAL MFF UK, Prague, Czech Republic, 2005. URL http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/en/m-layer/pdf/m-man-en.pdf.

Daniel Zeman. How Much Will a RE-based Preprocessor Help a Statistical Parser? In *Proceedings of the 7th International Workshop on Parsing Technologies*, Beijing Daxue, Beijing, China, 2001. Tsinghua University Press. ISBN 7-302-04925-4. URL http://ufal.mff.cuni.cz/~zeman/publikace/2001-01/pbml2001.ps.

Daniel Zeman. *Parsing with a Statistical Dependency Model*. PhD thesis, ÚFAL MFF UK, Prague, Czech Republic, 2005. URL http://ufal.mff.cuni.cz/~zeman/publikace/disertace/index.htm.

Daniel Zeman and Zdeněk Žabokrtský. Improving Parsing Accuracy by Combining Diverse Dependency Parsers. In *Proceedings of the International Workshop on Parsing Technologies (IWPT 2005)*, Vancouver, British Columbia, 2005. Association for Computational Linguistics. URL http://ufal.mff.cuni.cz/~zeman/publikace/2005-02/iwpt2005.pdf.

Appendix A

Values of some attributes

Value	Meaning
root	root of the sentence
atom	atomic node
coap	root node of a paratactic structure
list	root node of a list structure
fphr	node representing a foreign-language expression
dphr	node representing the dependent part of an idiomatic expression
complex	complex node
qcomplex	quasi-complex node

Table A.1: List of values of the `nodetype` attribute

Value	Meaning
#Cor	the controllee in control constructions absent at the a-layer
#EmpVerb	a verbal predicate absent at the a-layer
#Forn	the governing node of a foreign-language expression
#Gen	a general participant absent at the a-layer
#Neg	a syntactic negation
#Oblfm	an obligatory adjunct absent at the a-layer
#PersPron	personal or possessive pronouns (including the reflexives)
#QCor	a valency member in quasi-control constructions absent at the a-layer
#Rcp	a participant that were left out as a result of reciprocity

Table A.2: List of some special values of the `t_lemma` attribute

Value	Meaning	Czech example
Functors for the effective roots of independent clauses		
PRED	verbal clause	Pavel dal kytku Martině.
DENOM	nominal clause	Základní škola.
VOCAT	vocative clause	Hanko, podej mi to.
PARTL	interjectional clause	Hurá, vyhráli jsme!
PAR	parenthetic clause	Přijedu 13. prosince (pátek).
Argument functors		
ACT	actor	Otec pracuje.
PAT	patiens	Vaří oběd.
EFF	effect	Jmenovali ho předsedou.
ADDR	addressee	Poslal dárek příteli.
ORIG	origo	Vyrábí nábytek ze dřeva.
Temporal functors		
TWHEN	temporal: when	Přijdu zítra.
TFHL	temporal: for how long	Přijel na měsíc.
TFRWH	temporal: from when	Přeložil jednání ze soboty na dnešek.
THL	temporal: how long	Stihnul to za týden.
THO	temporal: how often	Pracuju na tom každý den.
TOWH	temporal: to when	Přeložil jednání ze soboty na dnešek.
TPAR	temporal: parallel	Během září ani jednou nepršelo.
TSIN	temporal: since when	Budu pracovat od zítra.
TTILL	temporal: till	Udělám to do pátku.
Locative and directional functors		
DIR1	direction: where from	Přijel z Prahy.
DIR2	direction: which way	Jdou lesem.
DIR3	direction: where to	Přišel domů.
LOC	locative	Pracuje v Praze.
Functors for implicational (causal) relations		
AIM	aim	Cvičí, aby zhubla.
CAUS	cause	Z důvodu nemoci zavřeno.
COND	condition	Když spí, nezlobí.
CNCS	concession	Navzdory svému věku je zdravá.
INTT	intent	Šel nakoupit.
Functors for expressing manner		
ACMP	accompaniment	tatínek s maminkou
CPR	comparison	víc než tisíc korun
CRIT	criterion/standard	Seřaď slova podle abecedy.
DIFF	difference	Je vyšší o dva centimetry.
EXT	extent	V nádobě je přesně litr vody.
MANN	manner	Mluví hlasitě.
MEANS	means	Píše perem.

Value	Meaning	Czech example
REG	regard	Vzhledem k počasí nelze nic plánovat.
RESL	result	Mluví tak potichu, že mu nerozumíme.
RESTR	exception	Kromě tebe tam byli všichni.
Functors for rhematizers, sentence, linking and modal adverbial expressions		
ATT	attitude	Je to samozřejmě pravda.
INTF	false (expletive) subject	Ono prší.
MOD	modality	Pracuje asi na půl úvazku.
PREC	refer. to preceding text	A pak odešel.
RHEM	rhematizer	Jen Karel odešel.
Functors for multi-word lexical units and foreign-language expressions		
CPHR	nomin. part of a predicate	mít plán
DPHR	depend. part of phraseme	křížem krážem
FPHR	foreign phrase	cash flow
Specific adnominal functors		
APP	appurtenance	můj hrad
AUTH	author	Nezvalovy verše
ID	identity	hrad Karlštejn
MAT	content of a container	sklenice vody
RSTR	restriction	velký dům
Functors expressing the relations between the members of paratactic structures		
ADVS	adversative relation	Viděl, ale neslyšel.
CONFR	confrontation	Pavel se zlepšuje, kdežto Jan má čtyřky.
CONJ	coordination/conjunction	Pavel a Jan
CONTRA	conflict	otec versus syn
CSQ	consequence	Pracoval špatně, a proto dostal výpověď.
DISJ	disjunctive relation	Pojedu já, nebo ty.
GRAD	gradation	Běžel, ba utíkal.
REAS	casual relation	Dostal výpověď, neboť pracoval špatně.
APPS	appositional structure	substantivum, neboli podstatné jméno
OPER	math. operation	pět až deset hodin
CM	conjunction modifier	otec a také syn
Other functors		
BEN	benefactive	Pracuje pro firmu.
COMPL	complement	Vrátila se unavená.
CONTRD	confrontation	Zatímco mzdy klesají, ceny se zvyšují.
HER	inheritance	šátek po matce
SUBS	substitution	Za otce jednal strýc.

Table A.3: List of functors

Value	Meaning
adj	adjective
adv	adverb
n	noun
v	verb
def/indef	definite/indefinite
demon	demonstrative
denot	denominating
grad/ngrad	gradable/non-gradable
neg/nneg	possible/impossible to negate
pers	personal pronoun
pron	pronomial
quant	quantificational

Table A.4: List of parts of values of the `gram/sempos` attribute

Value[1]	Meaning
A	adjective
AC	adjective, nominal form
C	numeral
C=	numeral, written using digits
C}	numeral, written using Roman numerals
Co	numeral, multiplicative indefinite
Cv	numeral, multiplicative definite
Db	adverb, not forming negation and degrees of comparison
Dg	adverb, forming negation and degrees of comparison
II	interjection
J,	conjunction, subordinate
*J^*	conjunction, connecting main clauses
N	noun
P	pronoun
R	preposition
RF	preposition, part of
TT	particle
V	verb
VB	verb, present or future tense
Vc	verb, conditional
Ve	verb, transgressive present
Vf	verb, infinitive
Vi	verb, imperative form
Vp	verb, past participle, active
Vs	verb, past participle, passive
Vt	verb, present or future tense, with enclitics
Z:	punctuation

Table A.5: List of some two-letter m-tags

[1]Numbers *1–7* denoting cases are left out from the second position, similarly with *X* denoting a tag missing from the dictionary.