A Grammar-Checker for Czech

Vladislav Kuboň, Tomáš Holan, and Martin Plátek

Table of Contents

Chapter 1

Introduction

One of the major problems of current computational linguistics is the lack of really practical applications. Among main reasons of this situation is the fact that there are many fields of natural language processing, where the effort spent on basic research is not justified by successful industrial applications. In most fields it is possible to achieve a dramatic improvement of speed, efficiency and quality of NLP systems with very simple means. Any additional efforts to improve the systems further are expensive with respect to funds, manpower and time.

One of the ways how to overcome this obstacle is to focus the linguistic research on those fields where simple means do not provide satisfactory results. Automatic grammar checking is one of these fields. This statement is even more true with respect to free word order languages. With growing degree of word order freedom the usability of simple pattern matching techniques decreases. In languages with such a high degree of word order freedom as in most Slavic languages (Czech, Polish, Russian Slovak etc.) the set of syntactic errors which may be detected by means of simple pattern matching methods is almost negligible.

The automatic grammar checking is also one of the areas in which the recent development of modern industrial products (text editors) created the demand from the side of the industry to get applications which will increase the usability of these products. The spelling checkers are already core parts of all text editors and the step towards grammar checkers or proof readers seems to be inevitable.

This technical report contains an overview of methods, tools and theories used in development of a pivot implementation of a grammar-based grammar checker of Czech. During the whole timespan of this project there were three main goals:

- linguistic research concerning the word-order properties of languages under consideration and also typical erroneous constructions which may be recognized by an automatic grammar checker;
- development of a method how to implement an efficient grammar checker; and
- pivot implementation of such a grammar checker proving that the method being developed may serve as a base for further, commercial exploitation.

All these main tasks were very closely related. In this report we would like to summarise the main ideas of our approach to the problem by concentrating on the second task, i.e. the description of a method how to build a practically oriented grammar checker for free-word order languages. We will not address the first task which had been already described in reports on linguistic problems of Czech.

Chapter 2

Parsing and grammar checking

It is quite common that some of the researchers working in the field of natural language parsing express the opinion that there is not a substantial difference between (robust) parsing and grammar checking. During our work in the last three years we have come to the conclusion that the nature of both tasks, although similar, contains also many different features and that it is necessary to develop special methods and tools in order to solve the problem of reliable, adequate, efficient and user-friendly grammar checker. The traditional methods of grammar checking (for example pattern matching or constraint relaxation techniques) and parsing do not provide a sufficient base for this task, especially when applied to languages with a high degree of word order freedom. That does not mean that they are not useful, it only means that they are too weak if they are used as the only means. Constraint relaxation technique is for example suitable for checking of agreement errors, patter

The most important part of our work on pivot implementation of a grammar checker is a general method how to build a grammar checker for free-word order languages. The core of our approach is a strategy to try to filter out unproblematic clauses or sentences with means as simple as possible in order to save the time and resources for more complicated cases.

1 Basic differences between parsing and grammar checking

The scope of natural language parsing is very broad. It covers the area from experimental small scale parsers developed as a support for some very sophisticated high-level linguistic theory to practically oriented systems with a stress on efficiency and speed, as for example in some systems designed for (machine aided) machine translation. On the other hand, grammar checking is a very particular field, where speed and efficiency are always among the criteria for the usefulness of the system.

Natural language parser is usually a system that either is completely hidden from the user or is designed to be used by a very specialized user - mostly a computational linguist, who needs as complex and precise results as possible and usually does not care about the speed or efficiency of the computation. The linguist also prefers a certain way of expressing results in the standard form which is common among the community of specialists (for example feature structures, dependency trees etc.). Such a form is usually very complex and contains also redundant data (copying information into heads of constituents in constituent trees) or data which are not relevant for a given purpose.

Grammar checkers are being developed in order to provide an ordinary user of some text editor with fast and reliable information about the type and location of grammatical (and/or stylistic) errors in the text. This information must be given in a form acceptable to the user.

The user should not be buried under a heap of messages - sometimes there are more possible sources of an error (either one violation of a syntactic rule causes more inconsistencies on the surface level or one surface syntax inconsistency may be caused by the violation of different syntactic rules). The checker should be able to order the messages (according to some formal criteria) to give the user only some messages.

Another basic difference between a parser and grammar checker consists in the size of resources used in both systems. While it is quite common to restrict the area of a parser to some task specific domain, which also leads to the use of a restricted domain specific dictionary, a grammar checker is always supposed to be able to cover unrestricted text. Similar observation holds with respect to the grammar coverage, but in this case the difference might not be so crucial as in the case of lexical coverage. Many of experimental natural language parsers have a broad coverage of the syntax of a given language, but very few of them cover a substantial part of the lexicon of that language.

Last but not least is the difference in what we expect to get from a parser and from a grammar checker. A syntactic parser is considered to work correctly only when it is able to provide the user with a complete set of syntactic structures representing all possible readings of a particular sentence, or just with a subset not containing readings which are inappropriate in the given context. On the other hand, the grammar checker should issue a reliable information about the type and location of a grammatical error(s). That means that for correct sentences we do not need a complete syntactic parsing, we do need only the result of a syntactic recognition.

We think that the above-mentioned differences justify the opinion that grammar based grammar checking must be handled by special means developed for that purpose.

2 Problems to be solved

Every attempt to create a working prototype of a grammar checker is confronted with a number of general problems of syntactic parsing, but also with a set of task-specific problems.

The crucial problem of any grammar checker is the decision about the strategy of identification and localisation of grammatical errors in a text. We have met in principle two strategies, one using the results of unsuccessful parsing (for example in a form of all edges of a chart parser) and trying to guess where the error might be, while the other strategy tries to locate the syntactic inconsistencies (syntactic inconsistency is a term used for the surface manifestation of an underlying error - one error may cause many syntactic inconsistencies) already in the course of the parsing process. We have chosen the second strategy because we think that it has certain advantages especially for free word order languages.

The second major problem is a formalism that would be able to provide adequate means for syntactic constructions typical for a given language, not only the correct ones, but also those describing the most frequent error types. It is of course possible to use one formalism for parsing and then to apply a special formalism for error localization to the (partial) results of the parsing. However, it seems obvious that having two different formalisms for syntactic parsing and for error checking in one system might be inefficient both from the point of view of building and debugging the grammar checker and also from the point of view of its overall performance and speed.

Another big challenge of grammar checking (and also of syntactic parsing in general) is the development, testing, debugging and maintaining of a large grammar. Many of natural language

parsing systems use different kinds of preprocessing. The main work must nevertheless still be done by one large grammar, in order to capture all possible parses, which might get lost if the grammar is divided into parts with no overlap of rules between them. This is also one of the reasons why building a large scale grammar is usually a one-man task. There is also another source of troubles hidden in a large scale grammar with broad coverage of syntactic phenomena of a given language, and that is the number of different correct parses. It unavoidably grows with the size of a grammar - when every new rule, covering some marginal phenomenon of a given language, is added to the grammar, the probability that this new rule will interact with other rules and create new readings of some sentences grows. This might be useful for pure syntactic parsing, but, as we have already mentioned above, for grammar checking it is enough to get just one parse for a syntactically correct sentence.

Building a large scale syntactic dictionary is also a very difficult task to solve. It might probably be easy for pattern matching based approaches to grammar checking, but for the approaches based on the lexical and syntactic information, the process of providing the dictionary of the system (used for example for spell-checking) with a reliable set of lexically syntactic data is very long and costly. Especially verbs with their rich valency frames are a real pain. It is clear that a large part of the work on syntactic dictionary of the system must be carried out by hand, but nevertheless it might be possible to automate the process by means of different kinds of machine-readable resources (tagged text corpora etc.) which would provide a basis for subsequent manual correction of results obtained by the (semi)automatic procedure. The problem of creating a large scale syntactic dictionary exceeds the frame of this project and as our experience with previous large scale natural language systems which were being developed in our institute (e.g. RUSLAN - Czech-to-Russian MT system by K.Oliva, see [4]) shows, it is clear that if we wanted to get a reasonable coverage of the lexicon of a particular language, we would have to join our efforts with an industrial partner.

The reliability of error messages is also one of the areas which might be problematic. The fact that grammar checker targets the broadest community of text editor users also means that it should be very carefull in error messages and warnings. We think that the user is more tolerant of errors in complicated sentences which were not discovered by the grammar checker (sometimes even the user is not able to find an error in such a sentence) than to error messages and warnings issued by mistake for simple correct sentences.

Chapter 3

Formal Description

Of the function and structure of the pivot implementation of a grammar-checker

In this section we introduce a class of formal grammars and some derived notions capable to describe typical surface syntactic inconsistencies (errors) in free word order languages. By means of these notions we characterize the tasks and the logical structure of the our pilot implementation of the grammar-checker for Czech.

The form of our explication will be as is usual in the theory of formal languages and grammars and we will try not to be too formal.

Direct paradigms for our grammars which we call *robust free-order dependency grammars (RFODG)* are the (commutative) CF-grammars (see [3]), dependency grammars (see [2]), and categorial grammars (see [1]).

RFODG is to serve for the description of surface syntax; it provides the base of the parsing with subsequent localization and evaluation of syntactic inconsistencies (errors). We assume that RFODG follows up with the lexical and morphological analysis being completed. Taking RFODG as the base we formulate the task of individual components of the grammar-checker and their mutual interaction.

The grammar-checker is composed of the following components:

a) lexical analysis

b) grammar-checking analysis by RFODG

c) evaluation

Component b) corresponds in principle an extended and sophisticated parsing. A new denotation was chosen so that we could talk about the usual (free) parsing and about the grammar-checking analysis at the same time.

Component c) corresponds an evaluation of the results of the grammar-checking analysis in order to estimate the localization of the relevant syntactic inconsistencies and to estimate the degree of their relevance.

1 Lexical analysis

We assume that the result of the lexical and morphological analysis for each word form is a finite set of symbols representing lexical and morphological properties of the word form. We further assume that formal syntax is connected with the lexical analysis in that the terminal symbols of the formal grammar accounting for syntax are the symbols representing lexical and morphological properties of the word forms.

Formally: Let us have a finite set of lexical categories (terminals) T and a set of word forms A.

We say that we have a lexical analysis L_a on A and T, if we have a function $L_a : A \to \mathcal{P}(T)$. $\mathcal{P}(T)$ means the set of subsets of T.

Lexical analysis L_a (L_a-analysis) of the word form $b \in A$ is defined as the pair $(b, L_a(b))$.

L_a-analysis of the string w of word forms $a_1 a_2 \dots a_n \in A^*$ is defined as the string of pairs $L_a(w) = (a_1, B_1)(a_2, B_2) \dots (a_n, B_n)$, where $L_a(a_i) = B_i$.

2 Robust Free-Order Dependency Grammars

The notion of RFODG is an enhancement of a free-order dependency grammar (see [11]). We suppose that RFOD-grammar is defined simultaneously with the corresponding lexical analysis.

There are the following types of classification of the set of symbols which are used by *RFODG*:
a) *terminals* and the other symbols (*nonterminals*)
b) *deletable* and *nondeletable* symbols
c) *positive* and *negative* symbols.

The sets under a) have the usual meaning and, furthermore, a) specifies the relation of RFODG to the given lexical analysis L_a. The terminals of RFODG are the lexical categories of L_a.

The sets under b) and c) serve for the classification and localization of syntactic inconsistencies.

Definition. Let L_a be a lexical analysis on A and T. *Robust free-order dependency grammar (RFODG)* is a 5-tuple (Nd, Dl, T, St, P), where Nd is the set of nondeletable symbols, Dl is the set of deletable symbols, and the union of Nd and Dl is denoted as V. T is the set of terminals ($T \subset V$), $V - T$ is the set of nonterminals, $St \subset V$ is the set of root-symbols (starting symbols), and P is the set of rewriting rules of two types of the form:

a) $A \to_X BC$, where $A, B, C \in V$, X is denoted as the subscript of the rule, $X \in \{L, R, LP, RP\}$, and if $X \in \{L, LP\}$, then $C \in Dl$, and if $X \in \{R, RP\}$, then $B \in Dl$;

b) $A \to B$, where $A, B \in V$.

We suppose that $V = V_p \cup V_n$, where V_p is the set of positive (correct) symbols, and V_n is the set of negative symbols (negative symbols serve also as error messages).

In the following explanation the *RFODG* is considered to be an analytic (recognition) grammar.

The occurrence of the letter L in the subscripts of the rules means that the first symbol on the right-hand side of the rule is considered *dominant*, and the other *dependent*.

The occurrence of the letter R in the subscripts means that the second symbol on the right-hand side of the rule is considered *dominant*, and the first one *dependent*.

If a rule has only one symbol on its right-hand side, we consider the symbol as *dominant*.

Applying such a rule we rewrite an occurence of the right-hand side symbol by the left-hand side symbol.

We work with the following restriction: If the dominant symbol of a rule is deletable, then also the symbol on the left-hand side of the rule is deletable.

A rule whose right-hand side contains two symbols is applied (for a reduction) in the following way:

The dependent symbol is deleted, and the dominant one is rewritten (replaced) by the symbol standing on the left-hand side of the rule.

The rules $A \rightarrow_L BC$, $A \rightarrow_R BC$, can be applied for a reduction of a string z to any of the occurrences of symbols B, C in z, where B precedes C in z.

The rules $A \rightarrow_{LP} BC$, $A \rightarrow_{RP} BC$ can be applied for a reduction of a string z to any neighbouring occurrences of symbols B, C in z, where B precedes C in z.

We introduce a notion of *Tree* by G. The *Tree* should map the essential part of a history of deleting dependent symbols and rewriting dominant symbols, performed by the rules applied. By means of *Tree* we can introduce another relevant notion of *dependency tree* and some (relevant) types of restrictions of applications of rules of G.

There are some differences between *Tree* and a standard derivation tree of CFG:

- A nonterminal (constituent) of a *Tree* may cover a discontinuous subset of the input sentence.
- The terminals can be used to create any type of nodes, not only the leaves.
- Each node has a fixed horizontal position, which is shared by exactly one leaf of the *Tree*. This property of *Trees* is used for the localization of syntactic inconsistencies in analyzed sentences.

A *Tree* will be a tree with a root which has two types of edges:

a) *vertical*: these edges correspond to the rewriting of a dominant symbol by the symbol which is on the left-hand side of the rule used. The vertical edge links the node containing the original dominant symbol with the node containing the symbol from the left-hand side of the rule used.

b) *oblique*: these edges correspond to the deletion of a dependent symbol. The node with the dependent deleted symbol and the node containing the symbol from the left-hand side of the rule used are linked by this edge. The oblique edges describe, in fact, the dependencies between the corresponding terminals.

The degree of branching of a $Tree$ is not greater than 2.

The nodes of *Trees* are introduced in such a way that the set of nodes of a *Tree* Tr is sufficient to represent the dependences in Tr.

a) A node U of a *Tree* Tr is a 4-tuple $[A, i, j, k]$, where $A \in V$ is the symbol in U, i, j are natural numbers, k is a natural number or 0. Number i is called horizontal index of U,

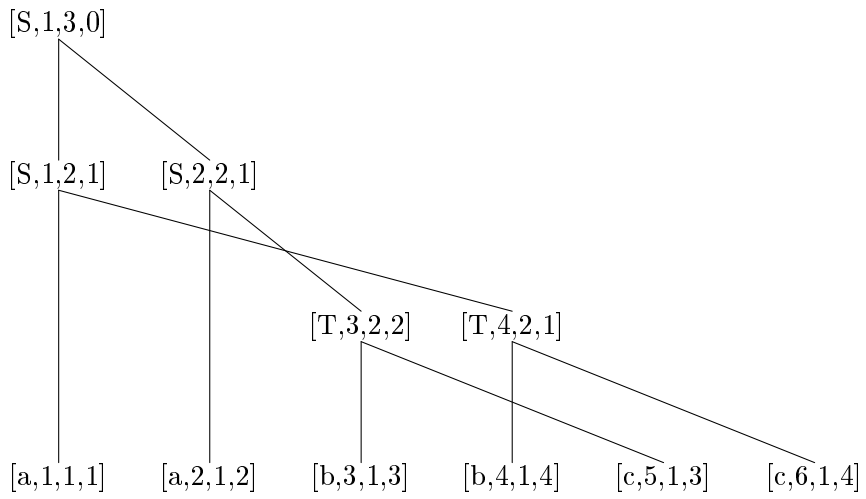

Figure 3.1: A *Tree* Tr_1 generated by the grammar G_1

j is called vertical index, k is called domination-index. The horizontal index expreses the correspondence of U with the i-th input symbol, the vertical index corresponds to the length of the path leading bottom-up from the leaf with the horizontal index i to U. The domination-index either represents the fact that any edge does not start in U, or it represents an edge starting in U.

b) Let $U = [A, i, j, k]$, U being a *Tree* Tr. If $j > 1$, then there is in Tr exactly one node U_1 of the form $[B, i, j-1, i]$. The pair (U_1, U) creates a vertical edge of Tr.

c) $U = [a, i, j, k]$ is a leaf of a *Tree* Tr, if and only if $a \in T$ and $j = 1$.

d) Let $U = [A, i, 1, k]$, U being a leaf of a *Tree* Tr. If $i > 1$, then in Tr there is exactly one leaf of the form $[c, i-1, 1, m]$.

e) The root of Tr is the single node with the domination-index equal to 0.

f) If $0 \neq k \neq i$ and a node U of the form $[A, i, j, k]$ is in Tr, then an oblique edge leads (bottom up) from U (dependent node) to its mother (dominant) node with the horizontal index k.

g) If a node U of the form $[A, i, j, i]$ is in Tr, then a vertical edge leads (bottom up) from U to its mother node with the same horizontal index i.

Example. Let the following grammar G_1 be a *RFODG*. $G_1 = (N_1, Dl_1, T_1, \{S\}, P)$, $T_1 = \{a, b, c\}$, $N_1 = \{a, b\}$, $Dl_1 = \{c, T, S\}$, $P = \{S \rightarrow_L aT|SS,\ T \rightarrow_L bc\}$. For the sake of simplicity we suppose that the corresponding lexical analysis is given in the following way: $\{(a, \{a\}), (b, \{b\}), (c, \{c\})\}$.

Conventions. $\mathcal{TR}(G)$ denotes the set of *Trees* rooted in a symbol from St, created by G. If $Tr \in \mathcal{TR}(G)$, we can say that Tr is *parsed* by G.

If $Tr \in \mathcal{TR}(G)$, and Tr contains only positive symbols (from V_p), we say that Tr is *positively parsed* by G. The set of positively parsed trees by G is denoted by $\mathcal{TR}_p(G)$.

If $Tr \in \mathcal{TR}(G)$, but $Tr \notin \mathcal{TR}_p(G)$, we say that Tr is *robustly parsed* by G.

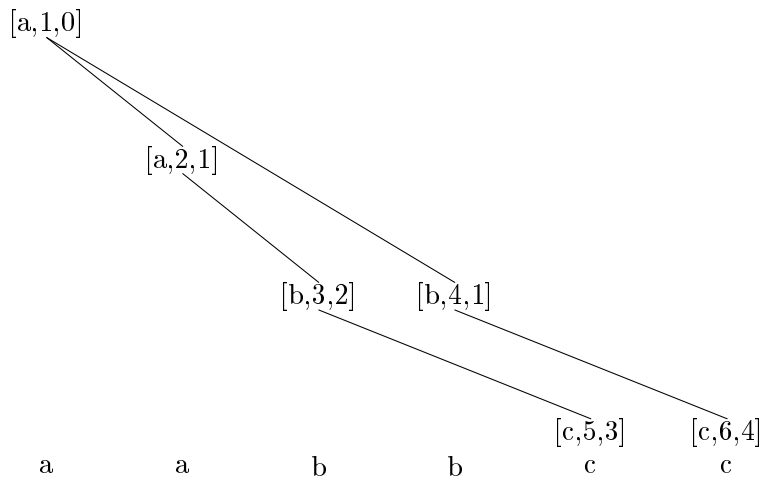

Figure 3.2: The dependency tree corresponding to the Tr_1

Now we introduce contracted trees and dependency trees parsed from a string of word forms. In fact, we get a dependency tree by the contraction of all the vertical edges of some *Tree* and by the substitution of symbols in the nodes by the related word form.

Definition. Let $Tr \in \mathcal{TR}(G)$. $Cn(Tr)$, the *contracted tree* of Tr, is defined as follows: The set of nodes of $Cn(Tr)$ is the set of 4-tuples $[a, b, i, k]$ for which there is a leaf u of Tr of the form $[a, i, 1, j]$, and the top node of the vertical path starting in u has the form $[b, i, m, k]$, where $i \neq k$. The edges of $Cn(Tr)$ correspond (one to one) to the oblique edges of Tr. They are fully represented by the domination-indices of nodes of $Cn(Tr)$.

We write $Cn(G) = \{Cn(Tr); Tr \in \mathcal{TR}(G)\}$.

Let L_a be the corresponding lexical analysis to G, $L_a : A \to \mathcal{P}(T)$. Let $Cn(Tr)$ contain exactly n nodes, and $w = a_1 a_2 \dots a_n$, $w \in A^*$, and $L_a(a_i) = B_i$, $b_i \in B_i$, and let $[b_i, c_i, i, k_i]$ denote the i-th node of $Cn(Tr)$ for $i = 1, \dots, n$. In such a case we say that the string w *is parsed into a dependency tree* $dT(w, Tr)$ by G. We write $w \in L(G)$. Thus $L(G)$ means the set of strings (sentences) parsed into some dependency tree by G.

It remains to define the dependency tree $dT(w, Tr)$ corresponding to the above assumptions. The set of nodes of the $dT(w, Tr)$ is the set of triplets $[a_i, i, k_i]$, where $[b_i, c_i, i, k_i]$ is a node of $Cn(Tr)$. The edges of the $dT(w, Tr)$ correspond (one to one) to the edges of the $Cn(Tr)$. That means if $k_i \neq 0$ (i.e. if the node is not the root) then there is an edge leading from the node $[a_i, i, k_i]$ to the node $[a_{k_i}, k_i, k_{k_i}]$.

We denote as $dT(w, G)$ the set of $dT(w, Tr)$, where w is parsed into $dT(w, Tr)$ by G. The $dT(G)$ denotes the union of all $dT(w, G)$ for $w \in L(G)$.

We can also write $\mathcal{TR}(w, G) = \{Tr; dT(w, Tr) \in dT(w, G)\}$. We say that $\mathcal{TR}(w, G)$ is the set of *trees* parsed from w. Similarly $Cn(w, G) = \{Cn(Tr); Tr \in dT(w, G)\}$.

We say that w is *positively parsed* by G if there is a positively parsed *Tree* in $\mathcal{TR}(w, G)$. $L_p(G)$ denotes the subset of $L(G)$ formed by the sentences positively parsed by G.

We say that a sentence w is *robustly parsed* if it is parsed, but not positively parsed.

We say that two trees (*Trees*, dependency trees, contracted trees) Tr_1, Tr_2 are structurally equivalent if it is possible to apply on the set of their nodes a one-to-one mapping f such that

from the $f(u) = v$ follows that the indices of u and v are identical. Thus u and v may differ only in the symbols they contain.

In the following subsections we are going to define several types of measures and limitations in order to be able to distinguish the free parsing from the grammar-checking analysis and to characterize different designs of the grammar-checking analysis.

2.1 Measures and restrictions on trees

In the following definition we introduce the notion of the *coverage* of a node of a *Tree*.

Definition. Let Tr be from $\mathcal{TR}(G)$. Let u be a node of Tr.

We denote as $Cov(u, Tr)$ the set of horizontal indices of nodes from which a path (bottom up) leads to u ($Cov(u, Tr)$ always contains the horizontal index of u). We say that $Cov(u, Tr)$ is the *coverage* of u (by Tr).

Example. The coverages of the nodes of Tr_1 from Figure 1 are shown in the following example:

$Cov([a, 1, 1, 1], Tr_1) = \{1\}$,
$Cov([a, 2, 1, 2], Tr_1) = \{2\}$,
$Cov([b, 3, 1, 3], Tr_1) = \{3\}$,
$Cov([b, 4, 1, 4], Tr_1) = \{4\}$,
$Cov([c, 5, 1, 3], Tr_1) = \{5\}$,
$Cov([c, 6, 1, 4], Tr_1) = \{6\}$
$Cov([T, 4, 2, 1], Tr_1) = \{4, 6\}$,
$Cov([T, 3, 2, 2], Tr_1) = \{3, 5\}$,
$Cov([S, 2, 2, 1], Tr_1) = \{2, 3, 5\}$,
$Cov([S, 1, 2, 1], Tr_1) = \{1, 4, 6\}$,
$Cov([S, 1, 3, 0], Tr_1) = \{1, 2, 3, 4, 5, 6\}$

Now we are going to define three (complexity) measures of non-projectivity, using the notion of coverage.

Definition. Let Tr be from $\mathcal{TR}(G)$. Let u be a node of Tr, $Cov(u, Tr) = \{i_1, i_2, \ldots, i_n\}$, and $i_1 < i_2, \ldots, i_{n-1} < i_n$. We say that the pair (i_j, i_{j+1}) forms a gap if $1 \leq j < n$, and $i_{j+1} - i_j > 1$. As $Ng(u, Tr)$ we denote the number of gaps in $Cov(u, Tr)$. $Ng(Tr)$ denotes the maximum from $\{Ng(u, Tr); u \in Tr\}$. We say that $Ng(Tr)$ is the *local number of gaps of* Tr.

Let (i, j),(k, l) be gaps of some nodes. If $i \leq k < l \leq j$ or $k \leq i < j \leq l$ holds, we say that (i, j), and (k, l) are *dependent*. The number of the maximal set of *independent gaps* of the nodes of the Tr is denoted as $Tng(Tr)$. We say that $Tng(Tr)$ is the *global number of gaps of* Tr.

We can easily see that $Tng(Tr) \geq Ng(Tr)$ for any Tr.

Definition. Let Tr be from $\mathcal{TR}(G)$. Let u be a node of Tr, $Cov(u, Tr) = \{i_1, i_2, \ldots, i_n\}$, and $i_1 < i_2, \ldots, i_{n-1} < i_n$. As $Sng(u, Tr)$ we denote the maximal number from the set $\{i_{j+1} - (i_j + 1); 1 \leq j < n\}$. $Sng(Tr)$ denotes the maximum from $\{Sng(u, Tr); u \in Tr\}$. We say that $Sng(Tr)$ is the *size of gaps of* Tr.

Example. Let us take again the *Tree* Tr_1 from previous examples. The following coverages contain a gap:

$Cov([T, 4, 2, 1], Tr_1) = \{4, 6\}$ has one gap (4,6),
$Cov([T, 3, 2, 2], Tr_1) = \{3, 5\}$ has one gap (3,5),
$Cov([S, 2, 2, 1], Tr_1) = \{2, 3, 5\}$ has one gap (3,5),
$Cov([S, 1, 2, 1], Tr_1) = \{1, 4, 6\}$ has two gaps (1,4) and (4,6).

We can see that Tr_1 has three gaps (1,4),(3,5),(4,6), and $Ng(Tr_1) = 2$, $Tng(Tr_1) = 3$, and

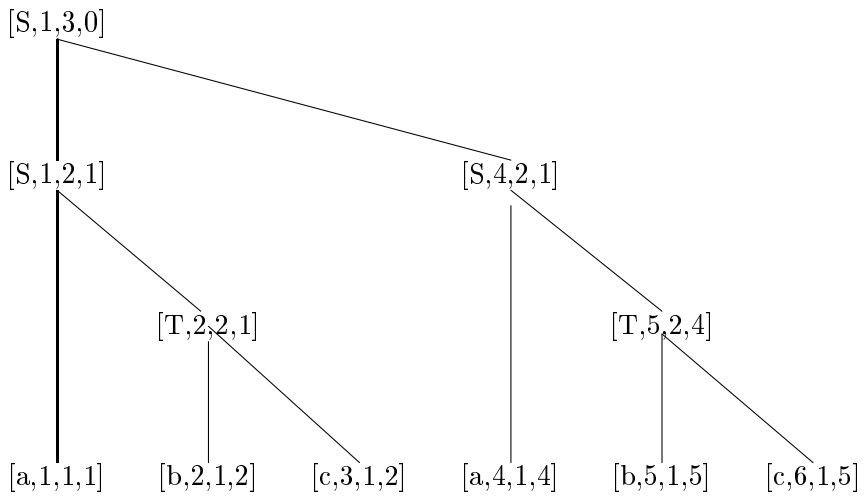

Figure 3.3: A projective *Tree* Tr_2 parsed by the grammar G_1

$Sng(Tr_1) = 2$.

Convention. If $Sng(Tr) = Tng(Tr) = Ng(Tr) = 0$, we say that Tr is *projective*. In the opposite case we say that Tr is *non-projective*.

Definition. Let $Tr \in \mathcal{TR}(G)$. We say that a node of Tr is negative, if it contains a negative symbol. As $Rob(Tr)$ we denote the number of negative nodes in Tr. We say that $Rob(Tr)$ is the *degree of robustness* of Tr.

Definition. Let us denote as $TR(w, G, i, j, k)$ the set of trees from $\mathcal{TR}(w, G)$ such that their value of the *Ng* function does not exceed i, the value of the *Rob* function does not exceed j, and the value of the *Tng* function does not exceed k.

2.2 Free parsing and grammar-checking analysis

An important idea of our approach to grammar-checking analysis may be demonstrated using an example of a Czech sentence. Let us take the sentence

"Kter dvata chtla dostat ovoce?"

(Word-for-word translation: Which girls wanted [to] get fruit?)

Some of the native speakers consider this sentence to be correct, some say that it is incorrect and most of them are uncertain, but they usually say that this sentence is neither correct nor incorrect, it is simply weird. The problem is that the sentence has at least two readings. One is correct (but non-projective), with the pronoun "Kter" [Which] depending on the noun "ovoce" [fruit]. The other reading contains a syntactic inconsistency between the pronoun "Kter" [Which] and the noun "dvata" [girls] - these two words disagree in gender.

A typical syntactic parser will prefer one of the possible readings of the above-mentioned sentence - probably the reading not containing an error. This behaviour is natural - if there is a correct parse, the parser should be able to find it and if it really finds it, then there is no reason to provide any other solution. A grammar checker, on the other hand, has to take into account that sometimes a sentence which is considered to be syntactically incorrect by most of native speakers may have a more or less obscure reading which is correct from the point of

view of formal syntax. Native speakers of a particular language usually take into account not only the formal syntax of the sentence when they decide about the grammatical correctness or incorrectness. Even in some grammar books syntactic rules are described by means of semantics, pragmatics or extralinguistic knowledge.

One of the problems of grammar checking lies in identification of improper combinations of verbs and prepositional phrases. A number of verbs requires one of its participants to be a prepositional phrase with a particular obligatory preposition. Since every preposition may serve also for prepositional cases of free modifiers (adjuncts), which are acceptable with any verb, it is impossible to check incorrect prepositional cases of participants on a purely syntactic basis. Almost every participant may be omitted in Czech (with respect to given context), so without a semantic or even pragmatic information it is not clear whether the particular sentence is correct or not. Let us illustrate these facts on the following examples (word-for-word translation):

Karel mu[dat.] pedstavil Milenu[acc.].

(Karel him introduced Milena - Karel introduced Milena to him)

Karel ho[acc.] pedstavil Milen[dat.].

(Karel him introduced [to] Milena - Karel introduced him to Milena)

**Karel ho[acc.] pedstavil Milenu[acc.].*

(Karel him introduced Milena - Karel introduced him Milena)

In this case the grammar checker is able to issue an error message that the sentence contains one extra object or one of the objects is in a wrong case. On the other hand, no such result is possible in the second example:

Shodli jsme se jen na[loc.] jedinm een.

[Agreed we ourselves only on one solution. - We agreed only on one solution]

Shodli jsme se jen na[acc.] chvli.

[Agreed we ourselves only for [a] while - We agreed only for a while]

**Shodli jsme se jen na[acc.] sestru.*

[Agreed we ourselves only for sister - We agreed only on a sister]

The third variant of this example cannot be distinguished from the second one by means of surface syntax only.

For us the only way how to solve this problem is to stay in the field of (surface) syntax and to try to use approaches slightly different from classical syntactic parsing in order to provide as much information about a particular input sentence as possible. By means of the defined complexity measures for *Trees* we are able to use a variety of settings in the course of grammar-checker analysis in order to capture the subtle differences between correct, incorrect and "weird" sentences.

Convention. As *free parsing* of a given string of word forms w we consider the computation of the set $\mathcal{TR}(w, G)$.

The experiments done with free parsing and with the grammar being developed have led us to the formulation of the following limitation:

In any alternative of the grammar-checking analysis we can confine ourselves to those computations of a tree for which the value of the Ng function is not greater than 1.

We give below three variants of the grammar-checking analysis. The three variants were created as the result of an endeavour to decrease the inadequate ambiguity of the free parsing and to formulate reasonable limitations for the degree of free word order in Czech and as the result of an endeavour to speed up the grammar-checking analysis.

The second phase in all three types of the grammar-checking analysis is formulated in such a way that we could account for the fact stressed in the previous section: it is often the case that the agreement errors in Czech sentences can manifest themselves as correct non-projective constructions. The major difference between the single variants consists in the formulation of their second phase, and third phase. We hope that the proposed experiments with the variants will show that the variants create an improving sequence regarding the adequatness and the speed of computation.

2.3 The first variant of the grammar-checking analysis

The first variant of the grammar-checking analysis is divided into three phases:

a) Positive projective

b) Positive nonprojective, negative projective

c) Negative nonprojective

In the first phase (a) , the given string w being analysed is tested whether the set $TR(w, G, 0, 0, 0)$, or, in other words the set of projective positively parsed trees, is empty. If this set is non-empty, the grammar-checking analysis ends, w is considered correct and the grammar-checker is no more preoccupied with w.

If the set $TR(w, G, 0, 0, 0)$ is empty, the second phase of the grammar-checking analysis starts.

In the second phase (b) , it is tested whether the set $TR2$, which means the union of the sets $TR(w, G, 1, 0, j)$ and $TR(w, G, 1, i, 0)$ over all natural numbers i and j, is nonempty. If $TR2$ is nonempty, it is handed over to be evaluated. In case that $TR2$ is empty, the third phase of the grammar-checking analysis starts.

In the third phase (c) , it is tested whether the set $TR3 = Tr(w, G)$ is nonempty. If $TR3$ is nonempty, it is handed over to be evaluated. If $TR3$ is empty, the message is handed over informing that the grammar-checking analysis failed.

The first variant of the grammar-checking analysis was implemented almost two years ago. We have a lot of experience with it. It allowed us to design the next two variants. We will see that they take over some tasks which according to the first variant should be done in the course of the evaluation.

2.4 The second version of the grammar-checking analysis

The second variant of the grammar-checking analysis is again divided into three phases a),b),c). The only difference from the previous variant lies in the second and third phase. Therefore we

omit here the description of the first phase.

The second phase b): Let min be the smallest number j such that $TR(w, G, 1, j, 0)$ is non-empty. If min exists, then in the second phase the set of trees $TR2$, is computed and handed over to the evaluation. In this case $TR2$ means the union of the set $TR(w, G, 1, min, 0)$ and of the sets of the form $TR(w, G, 1, 0, i)$ without any limitation on i. In case min does not exist, the third phase of the grammar-checking analysis starts.

Let m_3 be the smallest number i such that $TR(w, G, 1, i, j)$ is non-empty for some j. Then in the third phase the set of trees $TR3$ is computed and handed over for the evaluation. In this case $TR3$ means the union of the sets of the form $TR(w, G, 1, m_3, j)$.

If $Tr(w, G)$ is empty, the message is handed over that the grammar-checking analysis failed.

The second version uses the measure *Rob* for the formulation of its limitations. In the third version the measure *Tng* will be also used.

2.5 The third version of the grammar-checking analysis

The third variant of the grammar-checking analysis is again divided into three phases a),b),c), and the only differences from the previous variant are in the second and the third phases. Despite this fact we describe all three phases here.

In the first phase it is tested whether for the given analysed string w the set $TR(w, G, 0, 0, 0)$, or, in other words, the set of projective, positive trees, is empty. If this set is non-empty, the grammar-checking analysis ends, w is considered correct and the grammar-checker is no more preoccupied with w.

If the set $TR(w, G, 0, 0, 0)$ is empty, the second phase of the grammar-checking analysis starts.

Let min be the smallest number j such that $TR(w, G, 1, 0, j)$ is non-empty. If min exists, then in the second phase the set of trees $TR2 = TR(w, G, 1, i, j)$, where $i+j \leq min$, is computed and handed over to be evaluated. In case min does not exist, the third phase of the grammar-checking analysis starts.

Formulating the goals of the third phase we shall consider the lexicographical ordering on the pairs of numbers: Let (i, j) and (k, l) be the pairs of numbers. We shall write $(i, j) < (k, l)$, if $i < k$, or $i = k, j < l$.

Let (i_m, j_m) be the minimum pair such that $TR(w, G, 1, i_m, j_m)$ is non-empty. Then in the third phase the set of trees $TR3 = TR(w, G, 1, i_m, j_m)$ is computed and handed over to be evaluated. If the minimum pair does not exist, the message is handed over that the grammar-checking analysis failed.

We omit here the versions of grammar-checking analysis which used the meassure *Sng*. However, we want to realize some such versions in the future.

The disscussion in the next subsection serves to introduce concepts which are used to formulate the tasks of the evaluation module.

2.6 Components of syntactic inconsistencies

We assume that together with any RFOD-grammar G there is given a certain classification of its negative symbols. The following description of the classification is not formal. It should show the motivation for the design of the negative rules, and also its impact on the localization of the corresponding inconsistencies. We mainly want to stress a different nature of agreement inconsistencies compared to the other types of inconsistencies. We consider the agreement errors to be typical violations of the rules of grammar of Czech. That is why agreement errors are paid due attention even at this quite general level.

Classification of negative nonterminals

a) A negative nonterminal signals the assumption that no positive rule can be applied that would rewrite the nondeletable symbol (from Nd) which is dominated (being rewritten) by this negative nonterminal to some deletable symbol (from Dl). It is further assumed (to distinguish this case from the case c)) that the nonterminal does not signal an agreement error or an error in morphology. The expected correction is an addition of a dependent word form to the string being analyzed. In this case, one source of inconsistency is directly determined (i.e. the symbol which cannot be rewritten). Because of the free word order in Czech we consider the localization of the missing dependent word form as ungrounded.

b) A negative nonterminal signals the assumption that no positive rule can be applied that would delete a deletable symbol (from Dl) dominated by this negative nonterminal. It is again assumed (to distinguish this from the case c)) that this fact cannot be corrected by a mere change of morphology of the word forms in the sentence. The expected correction is a deletion of the subtree whose governing node is the positively deletable symbol.

c) A negative nonterminal signals an agreement inconsistency or some other inconsistency that can be corrected only by morphological changes of the word forms in the analysed string. We denote such a nonterminal as *mf-symbol*.

In the present section we deal only with the case c). We assume that we have such a grammar that uses morphological information as an integrating attribute, and that the result of the integration of the morphological information from various branches is their conjunction. On this assumption, the syntactic inconsistencies will be manifested by the mf-symbol which will appear, due to the stepwise integration of morphological features, in the node which will manifest itself as inconsistent with the node to which an edge leads. This node does not need to correspond at all to the word form which would contribute to this inconsistency. However, there is one certainty. Out of all nodes which contribute to the signalled inconsistency, there is a path leading up to the node which is assigned the pertaining mf-symbol.

The previous observation leads us to the following requirement which serves for the localization of the source of the signalled agreement inconsistencies. We assume that the set of rules is divided into two subsets: the rules that transfer and those that do not transfer the agreement information.

For the sake of a more exact characterization of our suggestions for localizing the sources of agreement inconsistencies we are going to introduce the following concepts.

Definition. We assume that the set of rules of the grammar G is divided into two subsets: morphologically sensitive (*mf-sensitive*) rules and morphologically non-sensitive (*mf-non-sensitive*) rules. The rules which delete negative symbols are classified as mf-non-sensitive rules.

The rule which has a mf-symbol on the left-hand side is classified as a mf-sensitive rule. We call the edges of the tree which come into being by the application of the mf-sensitive rules *mf-sensitive edges*, the other edges are *mf-non-sensitive*.

Let $w \in A^*$, and $Tr \in \mathcal{TR}(w, G)$. Let Tr be parsed robustly. Let u be the node Tr containing a mf-symbol. The *mf-component of the Tree Tr corresponding to u* will denote the subtree of Tr having the following properties:

a) it contains u and the edge which is incident to it (if it is not the root),

b) it contains all nodes from which a path leads to u via the mf-sensitive edges.

The *mf-component* of the $Cn(Tr)$ resp. $dT(w, Tr)$ is the subtree of the $Cn(Tr)$ resp. $dT(w, Tr)$ corresponding to the *mf-component* of the Tr.

We will say that G with the specified mf-symbols and mf-rules is *mf-consistent*, if for every $Tr \in \mathcal{TR}(w, G)$ and its mf-component *mfc* there exists $w_1 \in A^*$ such that there exists $Tr_1 \in dT(w_1, G)$ that is structurally equivalent to Tr, where Tr_1 differs from Tr in the symbols of the *mfc* component only. Tr_1 has only positive symbols in the nodes corresponding to *mfc*. Also w_1 and w differ only in the word forms corresponding to the nodes from *mfc* and, moreover, these word forms differ only in their morphology.

Remark. Our goal is a construction and debugging of an adequate mf-consistent grammar G (RFODG) for Czech with a special emphasis laid on an adequate agreement checking.

2.7 Evaluation

Evaluation is a part of the system following the grammar-checking analysis. In this subsection we describe a version of evaluation which would fit together with the second and third variant of the grammar-checking analysis. The implementation of a version fitting together with the first variant of the grammar-checking analysis is described in the next section.

The function of the evaluation module depends on which phase of the grammar-checking analysis it followed by.

If the grammar-checking analysis ends up after the first phase, the evaluation module is not invoked at all because the analysed string is considered correct.

If the grammar-checking analysis ends up after the second phase, then the analysis provides the sets of trees $TR2$. In this case, the first task of the evaluation module is to check whether it is possible for the non-projective positive trees be also considered as an expression of an error in agreement in the projective readings of the analysed string w. The evaluation selects from the set $TR2$ a subset of those trees which do not contain negative symbols other than mf-symbols. This set is denoted as *TR-mf*. The dependency trees corresponding to *TR-mf* with marked *mf-components* will be enumerated (so that they could be drawn). These trees contain possible agreement errors. In case *TR-mf* is empty, the evaluation will not return any warning, which means that the analysed string is considered to be correct.

If the grammar-checking analysis ends in its third phase, it provides at the output the set of trees $TR3$ for evaluation. If the set $TR3$ is not empty and is sufficiently small, the evaluation enumerates dependency trees containing marked negative nodes (the relevant node of the contracted tree carries the negative symbol) and marked mf-components. The set $TR3$ is considered small if its cardinality is not greater than e.g. 5.

If the cardinality of the set $TR3$ is greater, it is necessary to reduce the number of messages about possible grammatical inconsistencies and to order them in an adequate manner in order to provide the user with most probable error message first.

There are certain cases when the reduction of the number of error messages is quite safe. This is for example the case of possible compression of the information about the mf-components. For the dependency trees with the same structure (i.e. the same dependency trees which resulted from different *Trees*) it is possible to present *integrated mf-components*.

Definition. An edge of a dependency tree from $dT(w, G)$ is a *mf-marked edge*, if it is an edge of some mf-component of a dependency tree with the same structure (the requirement for the same structure can be omitted in another, less refined variant of integrated components).

For a dependency tree dT, its integrated mf-components are the maximal contiguous subtrees formed by mf-marked edges.

In case that the set $TR3$ or $TR2$ is big, the evaluation gives the sequence of dependency trees with integrated mf-components and with marked nodes, that do not belong to any integrated mf-component and therefore corresponds to some negative symbol other than a mf-symbol. The sequence of presented dependency trees is ordered according to the following rules. The priority of rule applications is given as follows:

a) The smaller the number of negative nodes not belonging to some integrated mf-component contained in Tr , the better is the position of Tr in the list.

b) The smaller number of integrated mf-components Tr contains, the better is the position of Tr in the list.

For the ordering of dependency trees we can consider even more sophisticated criteria. However, we have not specified them so far.

If the grammar-checking analysis ends up unsuccessfully, no tasks are performed by the evaluation module and the user receives a warning that the analysis failed. The evaluation of partial results is not yet implemented. Such an evaluation is sensible only with a lot of experience with a stabilized grammar tested on a representative, sufficiently large samples of texts. The evaluation of partial results without that experience may lead to incorrect conclusions.

It is, however, questionable whether after having a lot of experience it is not easier to complete the grammar so that it would cover all (theoretically possible) alternatives of input strings. However, this approach may be dubious because in this case the grammar will probably be too large and ambiguous. This would mean that the third phase of the grammar-checking analysis driven by such a grammar would be probably too lengthy.

3 Implementation of the evaluating module

The current implementation of a prototype of the module for error evaluation is realised as an independent Prolog program, which gets its input files in off-line mode from the previous phase of computation, i.e. from the first or the second version of the grammar-checking analysis.

The input: The results of the analysis of a single sentence are recorded in a file, which is the output of the grammar-checking analysis phase. The file contains all the items which have been derived for the given sentence. The items contain all usable information, especially the information about error edges with the identification (i.e. number) of the error.

The output: The error evaluation can end up with three different results:

a) *No errors:* The whole sentence was successfully parsed and there is at least one tree containing no error sign (negative symbol).

b) *No complete derivation:* If no tree for the whole sentence can be derived despite the application of negative rules, i.e. the analysis failed, the prototype does not issue any specific information about errors.

c) *List of errors:* Some derivation trees for the whole sentence were found, but each of them contains at least one error.

The cases a) and b) give no specific information and can be identified in the course of the previous phase of processing. Therefore it is not necessary to start the error evaluation.

Evaluation procedure:

From the set of complete trees found by the analysis, the trees with the minimal number of error edges are selected and the error edges are assigned the following information: kind of error, superordinate and subordinate words.

Observations, advantages and disadvantages:

– If the same error edge occurs in more trees which have a different structure, the edge is listed only once.

– In some cases, e.g. in coordination, techniques used in the prototype appeared to be too weak, the description of errors was too imprecise. From this it follows that stronger methods like a concept of the error component are necessary (see the previous section).

The first tests showed that the obtained descriptions of errors are relatively precise and enable the user a good localisation and identification of errors. In some cases more messages about one error are given.

Possible improvements of the prototype:

– The insertion of modules which should decrease the ambiguity of results by simulating the third version of the grammar-checking analysis.

– Despite the fact that the analysis failed as a whole in case b) it is sometimes possible to identify certain types of errors, i.e. the applications of the negative rules, in "sufficiently big" subtrees.

– The concept of an error component is not implemented in the prototype.

Chapter 4

Complex sentences vs. long phrases

Whenever we leave the ground of artificial examples of the type "John loves Mary" when writing grammars and building syntactic parsers of natural languages and whenever we start using the input data for example from some text corpora, we cannot avoid a wide range of problems which are not present in favourite "linguistic" examples. The specific type of problems depends on the type of the chosen text. It is clear that in technical texts there are specific syntactic problems different from those encountered in newspaper articles. Nevertheless there is one problem probably common to all types of text - sentences from the real text are substantially longer than hand-crafted examples often used by linguists.

With the growing length of sentences parsing will be more complex with respect both to the length of the processing and to the number of resulting syntactic structures. Let us demonstrate the problem on a sample sentence from the corpus of Czech newspaper texts from the newspaper Lidov noviny. Let us take the sentence:

"KDS nepedpokld spoluprci se stranou pana Sldka a nen pravdou, e pedseda kesanskch demokrat pan Benda v telefonickm rozhovoru s Petrem Pithartem prosazoval ing. Dejmala do funkce ministra ivotnho prosted."

(Word for word translation: "CDP [does] not suppose cooperation with party [of] Mister Sldek and [it] isn't true, that chairman [of] Christian democrats Mister Benda in telephone discussion with Petr Pithart enforced ing. Dejmal to function [of] minister [of] life environment.")

In this basic form of the sentence, which is an exact transcription of the text from the corpus, the processing takes 13,07s by the positive projective phase of our parser and it provides 26 different variants of syntactic trees. During the processing there were 2272 items derived. The testing of this sentence and also of all the following ones was performed on Pentium 75MHz with 16MB RAM (this data concern experiments with an older implementation of our parser).

Such a relatively large number of variants is caused by the fact that our syntactic analysis uses only purely syntactic means - we do not take into account semantics or textual or sentential context. This is clear especially when adding free modifiers into the tree of computation, where for instance free modifiers at the end of our sample sentence create a great number of variants of syntactic structures and thus make the processing longer and more complicated. In order to demonstrate this problem we will take this sentence and modify it trying to find out what the main source of ineffectiveness of its parsing is.

If we look more closely at the number of ambiguities present with individual words, we notice that the most ambiguous word is the word (abbreviation) "ing." This word form is the same in all cases, genders and numbers. If we substitute this abbreviation by the full form of the word ("inenra" [engineer - [gen.]]) we get the following results: the sentence is processed 8,95s, the number of variants decreases by four (22) and the number of derived items is, of course, also

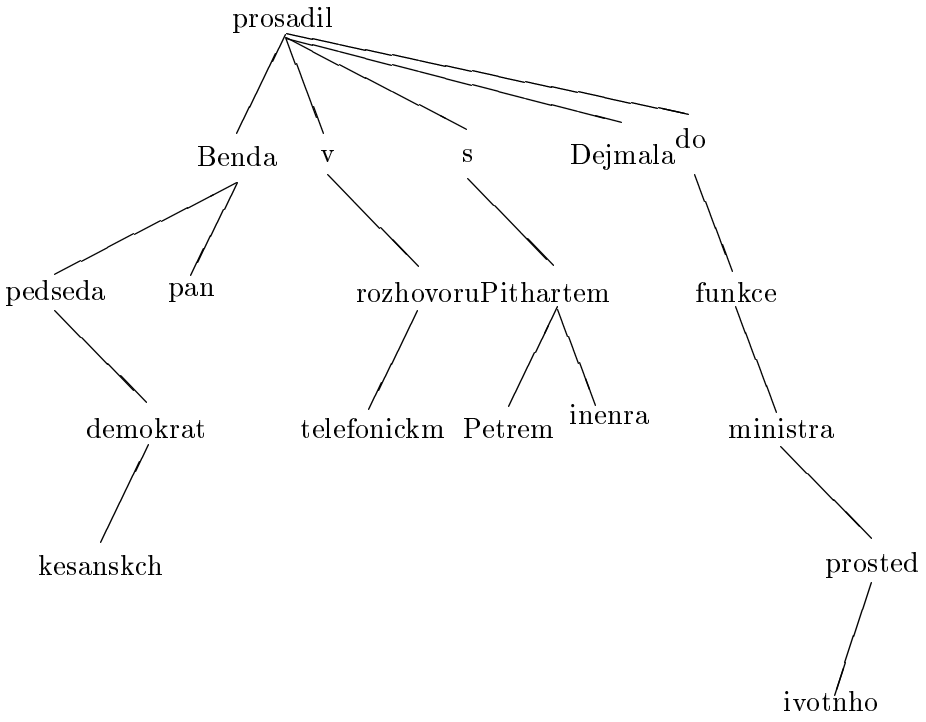

Figure 4.1:

smaller (1817). This speeding up would be even greater would we have worked with a negative or a nonprojective variant of the parser.

The next step is to delete another groups of words from the input sentence. Among the suitable candidates there is, for example, the prepositional phrase "v telefonickm rozhovoru" (in [the] telephone discussion). This phrase can be easily checked for grammatical correctness locally, because it has clear left and right borders (prepositions "v"and "s", respectively). We can easily solve here the usual problem of nominal groups, namely the problem where the nominal group ends on the right hand side. In general, we need to parse the whole sentence in order to get this information, but in some specific cases we can rely only on the surface word order.

After we had deleted this phrase, the processing time went down to 8,79s, the same number of syntactic representations as in the previous case was derived (22) and the number of items was slightly lower (1789). This phrase is therefore certainly not the main source of ineffectiveness in parsing. In order to speed up the processing even more we have to use another type of simplification.

The first step of simplifying the original input sentence represented almost 50a cosmetic change from abbreviation to full word form. From the point of view of localization of grammatical inconsistencies we can proceed even farther - the group title+surname in fact represents only one item; if we remove titles preceding surnames we do not change syntactic structure of the sentence. It is locally only a tiny bit simpler. When we look more closely at the resulting syntactic representation of the previous variants of the input sentence we may notice that the word "inenra" [engineer[gen.]] figures (inadequately, of course, in this case) also as a right-hand attribute to the word "Pithartem[instr.]", as it is shown in the figures 4,5 and 6 (for the sake of simplicity we demonstrate only the relevant part of derivation trees).

Let us remove the word "inenra" from the input sentence altogether. This time the processing time is only 3,74s, only 10 structures are created and 1021 items are derived. Another logical

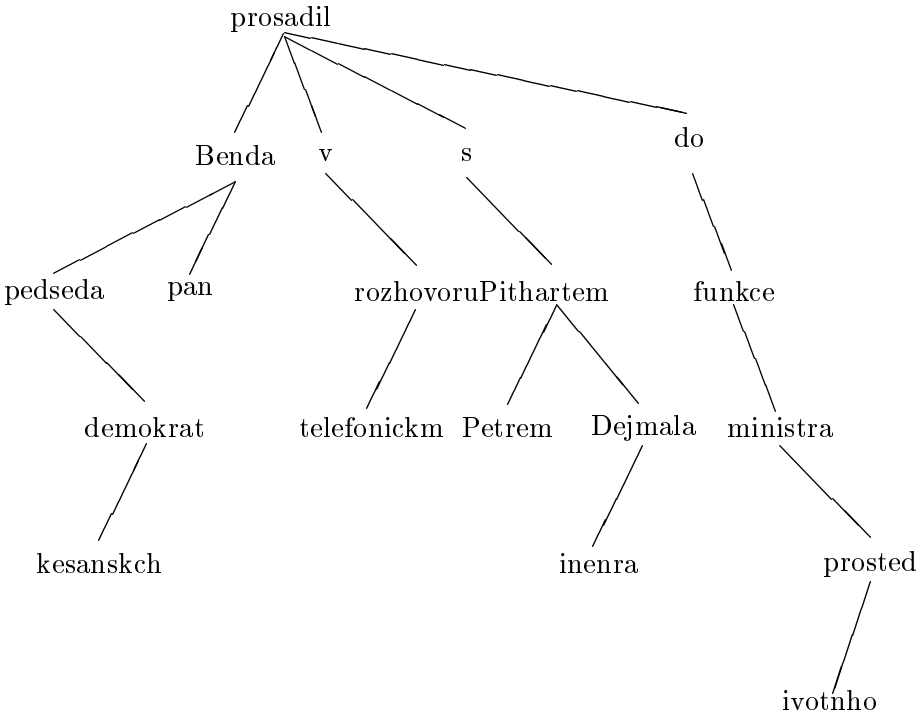

Figure 4.2:

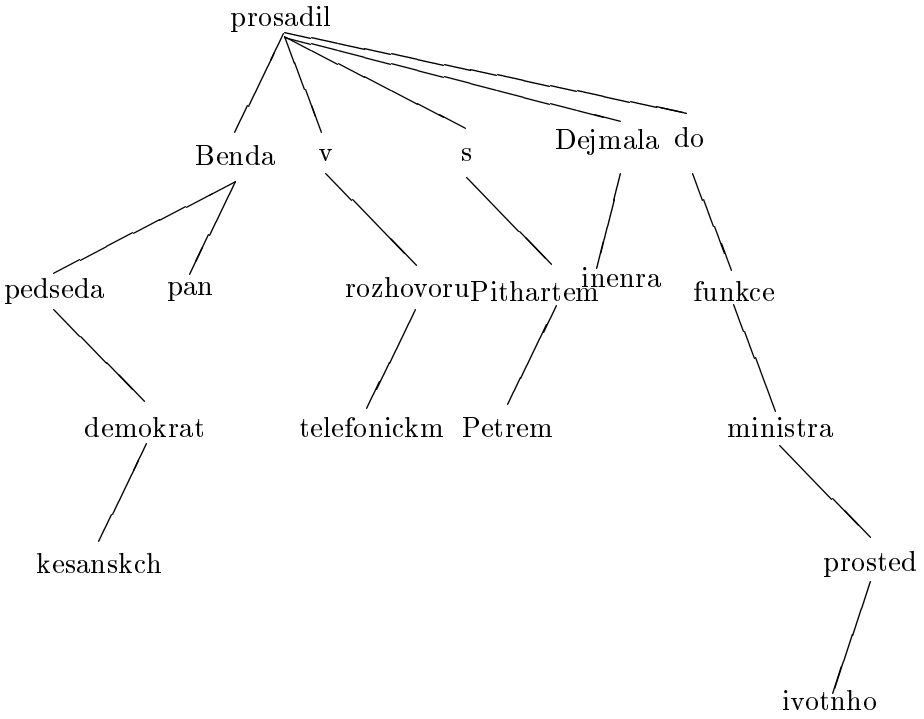

Figure 4.3:

step is to remove all other first names and titles which are placed immediately in front of their governing words. Those words are "pana" [mister [gen.]], "pan" and "Petrem". The claim that the first two words are unambiguous is supported by the fact that the form of the word "pn" [mister] is different in Czech in case the word is "independent" and in case it is used as a title (pna vs. pana [gen.,acc.], pn vs. pan[nom.]). When we make this change we gain more than 501,71s, also the number of resulting structures is a half of the original number (5) and only 587 items are derived. Another change we would like to demonstrate is the deletion of all other free modifiers the result of which is a certain "backbone" of the sentence.

After having carried out all deletions, we arrive at the following structure:

"KDS nepedpokld spoluprci a nen pravdou, e Benda prosadil Dejmala."

(Word for word translation: "CDP [does] not-suppose cooperation and [it] isn't true, that Benda enforced Dejmal.")

The result of the processing is a unique structure and 141 items are derived in 0,22s. The last variant of the input sentence will serve as a contrast to the previous ones. Let us take the last clause of the sentence, namely

"Pedseda kesanskch demokrat pan Benda v telefonickm rozhovoru s Petrem Pithartem prosazoval inenra Dejmala do funkce ministra ivotnho prosted."

["Chairman [of] Christian democrats Mister Benda in telephone discussion with Petr Pithart enforced ing. Dejmal to function [of] minister [of] environment.").

If we take into account the results of the previous examples we should not be surprised by the results. The processing time is 2,25s, 10 structures were created and 722 items were derived.

The conclusion of our series of examples is that from the point of view of the processing speed it is more important how many prepositional phrases the sentence contains, how these phrases are grouped together and whether they are separated by a verb, a relative clause etc. than how complex the sentence is. That gives us also a clear hint that the way to a substantial acceleration of processing by our "positive projective" grammar is to go through certain preprocessing of input sentences in order to remove those parts which may be involved in a syntactic error only locally and which slow down the processing by typical ambiguities.

Chapter 5

The Preprocessing Module

It is clear that a certain way of preprocessing of the input sentence may substantially accelerate its processing. One possible approach for deterministic preprocessing of the input was already developed in the frame of this project (see [9]) In this report we will try to look at this problem purely from the point of view of error localisation, because it will allow us to use strategies that are not "safe" from the point of view of full syntactic parsing.

If we take into account that for every grammar checker it is important to provide not only reliable error messages, but to issue the information for the user about the error contained in sentences in the text very quickly, we should try to comply with both requirements (antagonistic, as they may be, to a certain extent). We should let the user decide if he or she prefers more precise messages about errors, their type and location or if it suffices to perform a kind of "filtering" of the text which would for example mark sentences which are certainly correct. This possibility offers itself especially in connection with the fact that a number of typical and frequent grammatical errors in Czech texts is practically unidentifiable by an automatic grammar checker.

In our approach we try to create a certain set of means from the simplest and fastest to more complex and slow ones. The system will then be able to process simple and correct sentences relatively fast, indeed not at the price of making the processing of more complex sentences slower (for example for the reason that after failure of a simple computation the system would have to process the whole sentence by more complex means).

Chapter 6

The implementation

The implementation of our system was to a big extent influenced by the demand of effectiveness. For this reason we had to abandon even feature structures as a form of a representation of data. Our data structure is a set of attribute-value pairs with the data about valency frames of particular words as the only complex values (embedded attribute-value pairs).

An example of the representation of the Czech wordform "informoval" ([he] informed) follows:

```
informoval
lexf: informovat
wcl: vb
syntcl: v
v_cl: full
refl: 0
aspect: prf
frameset:
( [ actant: act  case: nom  prep: 0 ]
  [ actant: adr  case: acc  prep: 0 ]
  [ actant: pat  clause: z3e ]  )
neg: no
v_form: pastp
gender: ? inan , anim !
num: sg
END
```

The grammar of the system is composed of metarules representing whole sets of rules of the background formalism caled Robust Free Order Dependency Grammar (RFODG). The limited space of this technical note does not allow to present the description of RFODG here. The definition may be found for example in [1]. The metarules express a procedural description of the process of checking the applicability of a given metarule to a particular pair of input items A and B (A stands to the left from B in the input). In case that a particular rule may be applied to items A and B, a new item X is created. It is posssible to change values of the resulting item X by means of an assignment operator := . The constraint relaxation technique is implemented in the form of so called "soft constraints" - the constraints with an operator ? accompanied by an error marker may be relaxed in phases 2 and 3 ("hard constraints" with an operator = may never be relaxed).

The error anticipating rules are marked by a keyword NEGATIVE at the beginning of a rule

and are applied only in phases 2 and 3. The keyword PROJECTIVE indicates that the rule may be applied only in projective constructions.

An example of a (simplified) metarule describing the atachment of a nominal modifier in genitive case from the right hand side of the noun:

```
PROJECTIVE
   IF A.SYNTCL = n THEN ELSE
      IF A.SYNTCL = prep2 THEN ELSE FAIL ENDIF ENDIF
   B.SYNTCL = n
   B.case = gen
   A.RIGHTGEN = yes
   IF A.TITUL = yes THEN
      IF A.CASE = gen THEN
 IF A.GENDER = B.GENDER THEN
    IF A.NUM = B.NUM THEN FAIL ELSE ENDIF
ELSE ENDIF
      ELSE ENDIF
    ELSE ENDIF
   X:=A
   X.RIGHTGEN := no
   OK
   END_P
```

The interpretation of the grammar is performed by means of a slightly modified CYK algorithm (a description of this algorithm may be found for example in [6]). The grammar works with unambiguous input data (ambiguous words are represented as sets of unambiguous items). In order to speed up the checking of correct sentences we perform a syntactic recognition (not parsing) of the input sentence. For incorrect sentences both parsing and recognition of the input gives the same result, therefore this fact crea

tes no obstacle for the imminent application of "negative" rules (error anticipating rules and rules with relaxed constraints) after the first phase. All partial parses from the first phase are used in the second and third phases. For the purpose of testing and debugging the system we use full parsing even in the first phase.

A technical description of a working demo version of the whole system may be found in an appendix of this report.

1 Speeding up the performance

In nondeterministic parsers it is often the case that the author of a grammar has to prevent an unnecessary multiplication of results by means of "tricks" which are not supported by the linguistic theory - let us take for example the problem of subject - predicate - object construction. If we do not put any additional restriction on the order of application of rules then the rule filling the subcategorization slots for subject and object may be applied in two ways, either filling the slot for the subject first and for the object second or vice versa. Both ways create the same syntactic structure.

In such a case it is necessary to apply some additional constraints in the grammar - for example the restriction on the order of subcategorization (an item to the left of a verb should be processed first). This aproach makes the grammar more complicated than it is necessary and may also influence the quality of results (an error on the left hand side of a verb may also prevent an attachment of the items from the right hand side of the verb).

The interpreter of our grammar solves these situations itself. Every time a new item is created, the interpreter checks, if such an item with the same structure and coverage already exists. If yes, the new item is deleted.

This property of the interpreter is used together with other kinds of pruning techniques in all phases of grammar checking. Besides them there are also some techniques used especially in phases 2 and 3. The work with unambiguous input symbols allows fast parsing in the phase 1 (CYK is polynomial with respect to the length of the input), but creates some problems in the context of constraint relaxations used in subsequent phases. For example, a typical error in free word order languages is an eror in agreement. Let us suppose that we have the following three input words (the actual lexical value of these words may be neglected):

Preposition(accusative or locative) Adjective(Animate or inanimate gender, genitive or accusative sing.) Noun(Animate, genitive or accusative sing.)

These words represent $2 + 4 + 2 = 8$ unambiguous items. If we try to create a prepossitional phrase without constraint relaxation, we get one resulting item PP(Animate, accusative sing.). On the other hand after the relaxation of constraints there are 16 items created. One of them does not contain any syntactic inconsistency, remaining 15 have one or two syntactic inconsistencies. In a nondeterministic parser all 16 variants are used in the subsequent parsing. This causes a combinatorial explosion of mostly incorrect results.

There are two ways how to solve this problem. The first possible solution is to relax the constraints in certain order (to apply a hierarchy on constraints). We have chosen the other possible way, which prefers the subtrees with minimal number of errors. Every time a new branch or subtree is created, it is compared with the other branches or subtrees with the same structure and coverage and if it contains more errors than those already existing, it is not parsed further.

This technique substantially speeds up the processing of rules with relaxed constraints, but it has also one rather unpleasant side effect: the syntactic inconsistencies may be suppressed and appear later in a different location. This makes the task of the evaluating part of our system a bit more difficult, but nevertheless the gain on effectivity not accompanied by the loss of recall justifies the use of this technique.

The grammar of our system is implemented by means of metarules defining a set of rules for similar symbols, or some special composition of rules. The metarules describing possible erroneous constructions are treated similarly as those for positive constructions. They are incorporated into the grammar of the system either in a form of relaxed constraints (this kind of rules is typically used for capturing disagreement in gender, number and/or case) or in a form of special error handling rules. These metarules contain the keyword "NEGATIVE" and cover such phenomena as for example a missing comma before a relative pronoun.

The metarules containing the keyword "PROJECTIVE" can be applied only in a projective way (this corresponds to the rules of RFODG with the subscripts LP, RP).

The metarules are written in the form of a procedure. Such a procedure typically consists of the verification of all conditions of applicability of the rules, which are represented by this

metarule, and of the performance of the finally chosen single rule fulfilling all the conditions.

We adduce here a sample of a metarule implemented in our grammar:

```
;-------------------------------------------------------------
;  12.
;  twelfth rule - incongruent attribute in the genitive case from
;  the right
;
   PROJEKT TRUE
   A.SYNTCL = n
   B.SYNTCL = n


   B.case = gen
   A.RIGHTGEN = yes
   IF A.TITUL = yes THEN
      IF A.CASE = gen THEN FAIL ELSE ENDIF
ELSE ENDIF


   X:=A
   X.RIGHTGEN := no

   OK
   END_P
```

Chapter 7

Processing in layers

The layered architecture chosen as the basic means for localization of grammatical errors in our part of the project allows us to divide this process not only to three phases mentioned above, but it makes also a finer treatment of layers possible while the results obtained (items derived) in one layer are used also in the next layers. In this approach we are inspired by the work of Z.Kirschner ([5], [6]). A framework suitable for a formal description of this approach was designed in [10].

The basic idea is to divide the metarules of the grammar into certain groups in order to create, first, subtrees representing for example certain types of prepositional phrases, or more generally those parts of the sentence which may be involved only in some local error and therefore it is enough to check them only locally. It is clear that in this kind of processing we cannot avoid certain exceptions which will create errors in parsing. Thanks to the fact that in case of an unsuccessful result of parsing in layers there is still a possibility to build missing items in the following phases of syntactic parsing, we are not confronted with the number of problems which for example appear in the deterministic preprocessing module described in [9]. That means that there is a guarantee that the derived constructions will certainly be correct with respect to the grammar used; if this analysis does not fail then the result obtained is certainly one of the variants of the syntactic representation of the given sentence.

From the technical point of view we perform the processing of layers of metarules by means of a special data file, which contains the description of individual layers. At present we use 5 layers with the rules divided among them in the following way:

- 1st layer: *a metarule for processing titles and abbreviations preceding names*
- 2nd layer: *the metarule from the first layer together with metarules for processing prepositional and adjectival phrases*
- 3rd layer: *metarules from the previous layer together with metarules filling the valency slots and other metarules on the level of one clause*
- 4th layer: *metarules from the previous layer together with those processing complex sentences*
- 5th layer: *metarules for processing the left sentinel and the right hand side sentential border.*

In order to avoid a situation when at the level of (n+i)th layer suddenly a possibility (or even a necessity) appears to apply a metarule from the n-th layer and this situation in fact blocks the process of creating the tree of computation, we have the possibility of connecting certain layers

into a cycle, which is being performed till the new items are being derived. By means of this method it is possible to reduce drastically the number of resulting syntactic trees and also to shorten the time of computation. Let us demonstrate it on our sample sentence. The original variant of the testing sentence from the previous chapter is processed by means of the layers described above in 1.92s with 2 resulting syntactic structures and 455 derived items.

The application of layers may slow down the processing of short sentences (it has a fixed cost of opening the description file and consulting it during parsing process), therefore it is applied only to sentences longer than certain threshold (currently 15 words).

Another important point is the fact that the results of parsing in layers provide only positive information (i.e. the process is able to sort out sentences which are certainly correct, but the failure of parsing in layers does not necessarily mean that the sentence is incorrect). The same approach may not be used for error localization and identification, although the cases when parsing in layers fails on a correct sentence are quite rare.

On the other hand it is necessary to admit that the layers are useful only when applied to long sentences. If applied to sentences of the type "John loves Mary" the layers do not provide any time reduction. On the contrary, the processing time is a bit longer as a result of small time losses connected with the transition between layers. It is possible to solve this obstacle easily in the commercial version of the grammar checker by means of conditional treatment of sentences. That means that we would use layers only when processing sentences longer than a certain threshold number and thus the processing of shorter sentences will not be slowed down.

1 Suppression of variants

The division of the grammar into layers may substantially speed up the computation. There are of course other possible ways how to achieve the same goal, too. One of the possible solutions is to perform certain operations on the input sentence before it is processed according to the grammar of the system.

As was already described in the previous reports (see [12]), there are certain types of errors, which must be checked immediately after the morphological analysis when the sentence still is in the original shape, when no words are linked into syntactic structures and therefore it is possible to formulate conditions, for example, if two words are immediate neighbours. This is the question of errors in the vocalization of prepositions, but that is not the only case when it is useful to know the original word order.

Another area in which it is possible to a certain extent to simplify the processing of the sentence, is a temporary suppression of superfluous word meanings. As is shown for example in the report [9] , one of the major problems of preprocessing are ambiguous words which are applied incorrectly.

As an example we may use the sentence

"Kad vysok smluvn strana me pedloit prostednictvm generlnho tajemnka rady Evropy komisi kad dajn poruen ustanoven tto mluvy jinou vysokou smluvn stranou."

[Word for word translation - " Each high signatory party may submit *through the mediation of* general secretary [of] [the] Council [of] Europe each alleged violation [of] enactment [of] this declaration [by] other high signatory party"].

In this sentence from a European Union document the nominal group " Kad vysok smluvn strana" [Each high signatory party] is present twice. The problem is that both the pronoun "kad" [each] and the adjective "vysok" [high] are ambiguous, because they both may also fulfil the syntactic role of a noun ("high" in the sense "vysok zv" [deer] or "vysok kola" [university]). These nominal readings substantially raise the number of variants of syntactic representations of the given sentence. On the other hand, if we take into account that there is only one finite verb in the sentence it is clear that the sentence is not complex. This fact may be used for the reduction of ambiguities - the number of free nominal slots may be compared with the number of unambiguous nouns not preceded by prepositions and if the numbers agree, it is possible to suppress the nominal readings of adjectives and pronouns and to try to parse the sentence. In case we would like to proceed very carefully we can take into account also the possibility of a free modifier expressed by a nonprepositional instrumental case. In such a case we would not include nouns in instrumental case into the overall number of nouns unless the verb has a valency slot for a participant in instrumental case.

There is also a wide range of words which are theoretically ambiguous but the probability of their occurrence with one of the meanings is negligible. As an example we may present for instance the noun " ena" [woman], which may also be a transgressive of the verb " hnt" [drive]; the verb " li" [(they) went], which is also a dative, accusative or local form of the noun "le" [braces] and the preposition "podle" [according to] which is also an adverb [vilely].

From the point of view of preprocessing the most important groups are the groups of words starting with preposition and ending with a noun, between which there are only words with syntactic properties of an adjective (adjectives, pronouns, numerals). As we have already shown in the previous section, if we remove these words from the sentence successfully, we may substantially speed up the processing. One of the possible problems appears in case the given word form is ambiguous and may have a completely different roles in the sentence. We have tried to map this situation by analyzing a sample of texts contained in the Czech National Corpus. We have checked a few ambiguous word forms in order to illustrate the assumption that if a suitable context is taken into account, one or more of the readings of a given word form may be suppressed. We have chosen three word forms for this task: "se" [with/self], "podle" [according to/vilely] and "msto" [instead of/place]. The assumption that the simplest case is the preposition/adverb "podle" turned out to be correct. In the sample of 200 randomly chosen occurrences of this word in the context

```
"podle <succession of attributes> <noun in genitive case>"
```

199 cases were prepositions, only one occurrence was not clear. There was no adverb in this context. It seems that we may suppress the adverbial variant with a very high probability of success.

On the contrary, the word "msto" was chosen as an example of the word form where both readings are more balanced than in the previous case. This hypothesis was justified: out of 50 occurrences of this word form in the same context as in the previous case there were 17 prepositions, 28 nouns and in 5 cases there was a noun followed by another noun in the genitive case. Even worse, almost in all cases it was possible to decide to which category the word belongs only by means of the understanding of the meaning of the whole sentence. Typical occurrences of this type were for example the following ones: "Msto konn" [place/instead of performance],"...msto vkonu prce...", [place/instead of work], "...msto uren..." [place/instead of destination] etc.

The preposition/reflexive particle "se" was chosen due to the fact that there is a common

assumption that its syntactic role cannot be identified without a parsing of the full sentence. The results of our investigation were surprising. The assumption of the insolvable ambiguity is not correct, since out of 240 occurrences there were 211 reflexive particles, 23 occurrences in which it was possible to identify the word as preposition on the basis of the local context and only in 6 cases the decision made on the basis of a local context led to a wrong result when the particle was identified as a preposition (for example in cases "...zkrt se tm..." [it is shortened by this], "...zabvajc se zahraninmi vztahy..." [preoccupied by international relations] or "...stali jsme se hlavnm dodavatelem..." [we became the main supplier]). As speakers of Czech may already have noticed, it would be enough to use simple rules for vocalisation of prepositions to realize that only one occurrence is truly ambiguous - "...zabvajc se zahraninmi vztahy..." but it may be solved in the context of the whole sentence on the base of the fact that the verb "zabvat se" requires the presence of the reflexive particle in the sentence.

The above-mentioned simple rules for preprocessing are only a sample of a wide variety of possibilities. We have made only first few steps towards the development of a more representative set of preprocessing rules. The future work should concentrate on two main topics:

1. Development of a formalism capable to express preprocessing rules. The formalism used in the phase of surface syntactic analysis does not provide some of the means required in the preprocessing phase. For example, it does not have the means for expressing the conditions for a broader context or for the interaction of more than two input elements in one rule. It also seems to be a good idea to make the preprocessing formalism more deterministic than the formalism for syntax. It should be able to express such (meta)rules as for example:

```
Check whether any word in the input sentence has a valency slot
requiring the noun to be in the genitive.

If not, add the  (meta)rule no. 21 to the second layer
of   rules in the file VRSTVY.DAT.
or
Find a preposition and check whether it is followed
by items with SYNTCL = adj (optional)
and an item with SYNTCL = noun.

Check the group on the agreement in case of a preposition
and for the agreement in gender and number among the items
following the preposition.

If everything agrees, check whether the group is followed
by a full stop, unambiguous verb or preposition, comma,
question mark, connective or adverb.
If yes, mark the whole group
as correct and remove it from the input.
```

2. Thorough linguistic investigation of the corpus concentrating on the problem of identification and localization of groups of words which might provide reliable syntactic information or which might be deleted from the input sentence in order to speed up the processing of that sentence.

Both tasks will require a large amount of work. It opens a whole new area especially for the linguistic research, which will be useful not only for the development of new versions of a

grammar checker, but also for a syntactic parsing as such. The information about "syntactically unambiguous islands" in the sentence may lead to a new generation of parsers, which will combine deterministic and nondeterministic methods in order to get most accurate syntactic representation of the input sentence faster than by means of traditional methods.

Chapter 8

Conclusion

In this report we have summarized the results of research carried on in the frame of the project *Language Technology for Slavic Languages*. We believe that the main goal of the project, which was to develop a method for grammar based grammar checking of free word order languages and to create a pilot implementation of the grammar checker for Czech, has been achieved.

The method described in this report provides a base for future research in the field of grammar checking and opens a number of questions which will require further investigation. Some of the solutions are briefly sketched here, some will still have to be designed. The number of interesting problems we have encountered in the course of the work on this project makes us feel that we have touched a field worth investigating and that the effort spent on the project has brought useful results.

Bibliography

[1] Y. Bar-Hillel, C. Gaifman, F. Shamir: On categorial and phrase structure grammars, Bulletin of the Research Council Israel,F9, 1960, pp. 1-16

[2] D. G. Hays : Dependency theory: A formalism and some observation, Language 40, 1964, pp.511-514

[3] D.T.Huynh: Commutative Grammars: The complexity of Uniform word Problems, Information and Control 57, 1983, pp. 21-39

[4] K.Oliva : A parser for Czech Implemented in Systems Q, In: Explizite Beschreibung der Sprache und automatische Textbearabeitung, MFF UK, Prague, 1989

[5] Z. Kirschner: Private communications, 1994

[6] Z. Kirschner: CZECKER - a Maquette Grammar-Checker for Czech, The Prague Bulletin of Mathematical Lingistics 62, MFF UK Prague, 1994, pp. 5 - 30

[7] J.Panevov : Valency Frames and the Meaning of the Sentence, In: The Prague School of Stuctural and Functional Linguistics, Linguistic and Literary studies in Eastern Europe 41, ed. P.A. Luelsdorff, John Benjamin Publishing Company, 1994, pp. 223 - 244

[8] N. Sikkel: Parsing Schemata, Proefschrift, Enschede,1993

[9] J.Hric , A.Rosen, M. Strakov: Deterministic machine - report on the deterministic parsing moduleof the Grammar-Checker , Research Report in Joint Research ProjectPECO 2824, 1994

[10] M.Pltek : The Architecture of a Grammar Checker, In: Proceedings SOFSEM '94, Milovy, 1994, pp. 85-90

[11] T.Holan,V.Kuboň, M.Pltek: An Implementation of Syntactic Analysis of Czech, In: Proceedings of IWPT' 95, Charles University Prague, 1995, pp. 126-135

[12] Avgustinova, T., Bmov, A., Hajiov, E., Oliva, K., Panevov, J., Petkevi, V. (ed.), Sgall, P. and Skoumalov, H.: Linguistic Problems of Czech. Final Research Report for the JRP PECO 2824 project. Prague, 1995 (pp. 157)

The ÚFAL Technical Report Series

ÚFAL

ÚFAL (*Ústav Formalni a Aplikované Lingvistiky*) is the institute for formal and applied linguistics, at the Department of Mathematics and Physics of Charles University, Prague Czech Republic. ...

Technical Reports

The ÚFAL technical report series has been established with the aim of disseminate topical results of research currently pursued by members, cooperators, or visitors of the institute. Since November 1996, the following reports have been published:

TR-01 Eva Hajičová, *A History of Computational Linguistics in the Czech Republic*
Jan Hajič and Barbora Hladká, *Rule-Based Morphological Analysis*

TR-02 Vladislav Kuboň, Tomáš Holan, and Martin Plátek, *A Robust Grammar-Checker for Czech*

Further Information

Further information concerning the activities of ÚFAL should be directed to

Prof. Eva Hajičová
ÚFAL MFF UK
Malostranské náměstí 25
CZ-118 00 Praha 1, Czech Republic
⟨hajicova@ufal.mff.cuni.cz⟩
++420-2-2191-4253 (phone)
++420-2-2191-4309 (fax)