A Prototype of a Grammar Checker for Czech[i]

Tomáš Holan
Dept.of Software and Computer Science Education
Charles University, Prague,
Czech Republic
holan@ksvi.ms.mff.cuni.cz

Vladislav Kuboň
Inst.of Formal and Appl.Ling.
Charles University, Prague,
Czech Republic
vk@ufal.ms.mff.cuni.cz

Martin Plátek
Dept.of Theoretical Comp.Sc.
Charles University, Prague,
Czech Republic
platek@kki.ms.mff.cuni.cz

Abstract

This paper describes the implementation of a prototype of a grammar based grammar checker for Czech and the basic ideas behind this implementation. The demo is implemented as an independent program cooperating with Microsoft Word. The grammar checker uses specialized grammar formalism which generally enables to check errors in languages with a very high degree of word order freedom.

Introduction

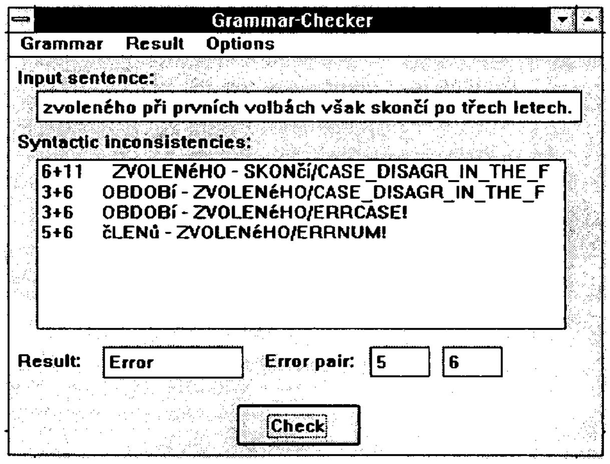
Automatic grammar checking is one of the fields of natural language processing where simple means do not provide satisfactory results. This statement is even more true with respect to grammar checking of the so-called free word order languages. With the growing degree of word order freedom the usability of simple pattern matching techniques decreases. In languages with such a high degree of word order freedom as in most Slavic languages the set of syntactic errors that may be detected by means of simple pattern matching methods is almost negligible. This is probably one of the reasons, why even though the famous paper [CH83] was written as long as 13 years ago, there are still very few articles about this topic, except papers like [K94] or [M96] which appeared only during the last three years.

In the present paper we describe the basic ideas behind an implementation of a prototype of a grammar checker for Czech. During the development of this application we had to solve a number of problems concerning the theoretical background, to develop a formalism allowing efficient implementation and of course to create a grammar and define the structure of the lexical data. The last but not least problem was to incorporate the prototype into an existing text editor.

How does the system work

In order to demonstrate the function of the pivot implementation of our system we decided to connect it to a commercially available text editor. We intended to create a DLL library with the standard grammar checking interface required by a particular text editor. This idea turned out to be unrealistic because the necessary interface is among the classified inside information in most companies. Fortunately there is the possibility to use a concept of Dynamic Data Exchange (DDE) for the communication between programs in the Microsoft Windows environment. This type of connection is of course much slower than the intended one, but for the purpose of this demonstration the difference in speed is not so important.

Our system can work with any text editor under Windows that contains a macro language supporting the DDE connection. For the purpose of the pivot implementation of the system we have chosen Microsoft Word 6.0. The grammar checker is implemented as an independent Windows application (GRAMMAR.EXE) which runs on the background of the Word. In order to be able to use GRAMMAR.EXE, we had to create a macro Grammar, assigned to the Grammar Checker item in the Tools menu. This macro selects a current sentence, sends it to GRAMMAR.EXE via DDE, receives the result and indicates the type of the result to the user. This activity is being performed for all sentences in the selection or for all sentences from the position of the cursor till the end of document.



Input sentence:
zvoleného při prvních volbách však skončí po třech letech.

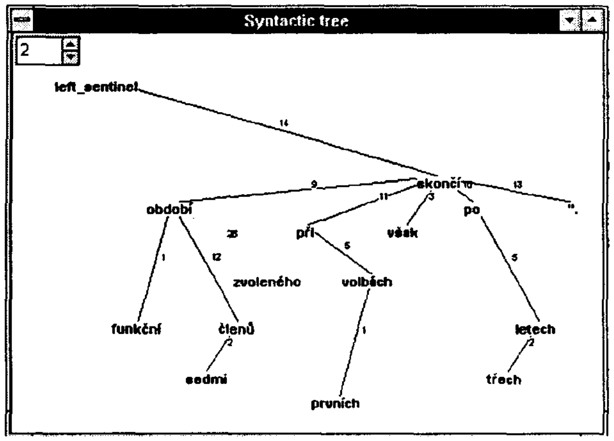
Syntactic inconsistencies:
```
6+11    ZVOLENéHO - SKONčí/CASE_DISAGR_IN_THE_F
3+6     OBDOBí - ZVOLENéHO/CASE_DISAGR_IN_THE_F
3+6     OBDOBí - ZVOLENéHO/ERRCASE!
5+6     čLENů - ZVOLENéHO/ERRNUM!
```

Result: Error   Error pair: 5   6

The user may get several types of messages about the correctness of the text:

a) The macro changes the color of words in the text according to the type of the detected error - the unknown words are marked blue, the pairs of words involved in a syntactic error are marked red.
b) The macro creates a message box with a warning each time there is an undesired result of grammar checking — either there was no result or the sentence was too complicated.
c) In case that the grammar checker identified and localized an error, it creates a message box with a short description of the error(s).

Because the grammar checker is running as an independent application, the user may also look at the complete results provided by it. When a message box containing an error message appears on the screen, the user may switch to GRAMMAR and get an additional information. The main window of GRAMMAR is able to provide either the complete list of errors, the statistics concerning for example the number of different syntactic trees built during grammar checking or even the result in the form of a syntactic tree. We do not suppose that the last option is interesting for a typical user, but if we do have all this information, why should we throw it out?



The architecture of the system

The design of the whole system is shown in the Fig.1. The grammar checker is composed basically of three parts:

1.Morphological and lexical analysis

This part is in fact an extended spelling checker. The input text is first checked for spelling errors, then the lexical and morphological analysis creates data, which are combined with the information contained in a separate syntactic dictionary. It would of course be possible to use only one dictionary containing morphosyntactic information about particular words (lemmas), but for the sake of an easier update of information during the development of the system we have decided to keep morphemic and syntactic data in separate files.

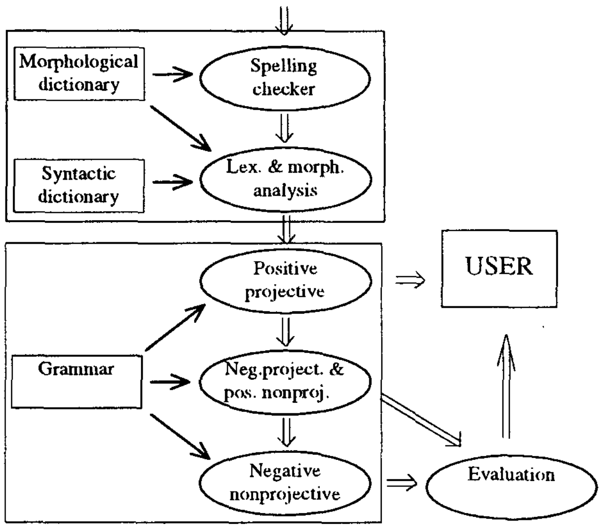


Fig 1:The architecture of the system

2.Grammar checking (extended variant of syntactic parsing)

This is the main part of the system. It tries to analyze the input sentence. There are three possible results of the analysis:

a) The analysis is successful and no syntactic inconsistencies were found (at this stage of processing it is too early to use the term syntactic error, because in our terminology the term error is reserved for something what is being announced to the user after the evaluation) — in this case the sentence is considered to be correct and no message is issued.
b) The analysis is successful, but all results contain at least one syntactic inconsistency. In this case it is necessary to pass the results to the evaluation phase.
c) The analysis fails and (probably for the reason of the incompleteness of the grammar) it cannot say anything about the input sentence. In such a case no error message is issued. We do not use any partial results for the evaluation of the possible source of an error. Partial results are misleading, because it is often the case that the error is buried somewhere inside the partial tree and no operations performed on partial trees can provide a correct error message. Besides that operations on (hundreds or thousands)

partial trees are very ineffective and they can also slow down substantially the processing of the given sentence.

3.Evaluation

This phase takes the results of the previous phase in the form of syntactic trees containing markers describing individual syntactic inconsistencies. It tries to locate the source of the error using an algorithm that compares available trees. According to the settings given by the user the evaluation phase issues warnings or error messages.

The core of the system is the second, grammar checking phase, therefore we will concentrate on the description of that phase.

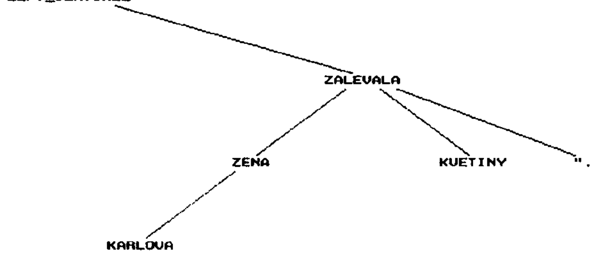
Process of grammar checking

The design of our system was motivated by a simple and natural idea — the grammar checker should not spend too much time on simple correct sentences. The composition of a grammar checking module tries to stick to this idea as much as possible. The processing of an input sentence is divided into three phases:

a) Positive projective

This phase is in fact a standard parser — it checks if it is possible to represent a given input sentence by means of a projective syntactic tree not containing any negative symbol (these symbols represent the application of a grammar rule with relaxed constraints or an error anticipating rule). If the answer is positive, the sentence is considered to be correct and no error message is issued.

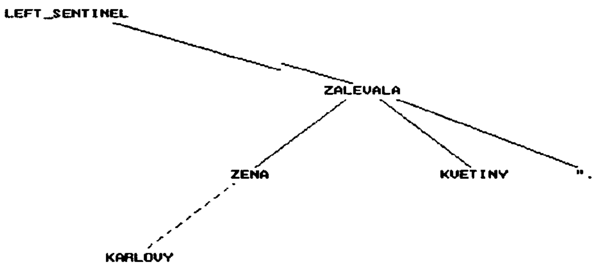
As an example we may take the following simple sentence: "Karlova žena zalévala květiny." (Word for word translation: Charles'[fem.sing] wife watered therefore its processing ends here. The system recognizes the structure of this sentence in the following way:



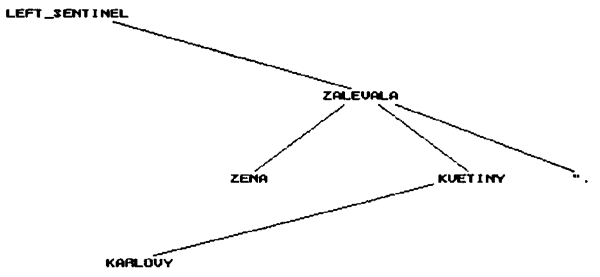
b) Positive nonprojective & negative projective

This phase tries to find a syntactic tree which either contains negative symbols or nonprojective constructions. A nonprojective subtree is a subtree with discontinuous coverage. It is often the case — for example in wh-sentences — that the sentence may be considered either syntactically incorrect or nonprojective — see examples in [COL94]. If such a syntactic tree exists, the evaluation phase tries to decide if there should be an error message, warning or nothing.

Let us present a slightly modified sentence from the previous paragraph: "Karlovy žena zalévala květiny." (Word for word translation: Charles'[fem.pl.] wife watered flowers). This sentence is ambiguous, it is either correct and nonprojective (meaning: Woman watered Charles' flowers) or incorrect (disagreement in number between "Karlovy" and "žena") and projective. Both results are achieved by this phase of the grammar checker:



Projective reading contains an error



Nonprojective reading

c) Negative nonprojective

Both nonprojective constructions and negative symbols are allowed. If this phase succeeds, the evaluation module issues a relevant error message or warning. In case that neither phase provides any result, no error message is issued. In case that the user wants to know which sentences were not analyzed properly, s/he may obtain a warning.

Although this division into phases worked fine for short sentences (for the sentences not more than 15 words long the first phase usually took about 1 second on Pentium 75 MHz), long and complicated sentences were unacceptably slow (even tens of seconds). These results turned our attention to the problem how to speed up the processing of correct sentences even further.

With the growing length of sentences the 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 Lidové noviny. Let us take the sentence:

"KDS nepředpokládá spolupráci se stranou pana Sládka a není pravdou, že předseda křesťanských demokratů pan Benda v telefonickém rozhovoru s Petrem Pithartem prosazoval ing. Dejmala do funkce ministra životního prostředí."

(Word for word translation: "CDP [does] not suppose cooperation with party [of] Mister Sládek 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] environment.")

In this basic form of the sentence, which is an exact transcription of the text from the corpus, the processing by the positive projective phase of our parser takes 13,07s 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.

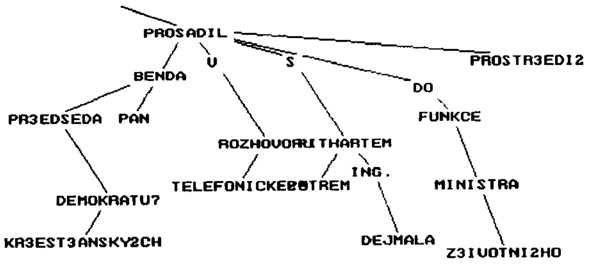
Such a relatively large number of variants is caused by the fact that our syntactic analysis uses only purely syntactic means - we do not take into account either semantics or textual or sentential context. That is the reason why free modifiers at the end of our sample sentence create a great number 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.

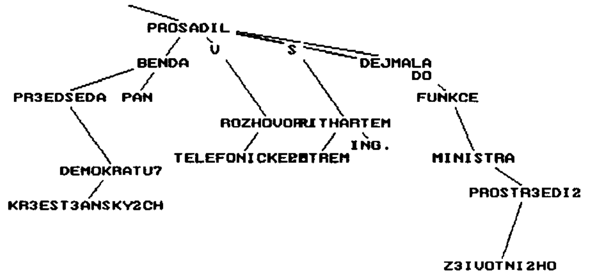
If we look more closely at the number of ambiguities present with individual words, we notice that the most ambiguous word is the word (abbreviation) "ing." This word form is the same in all cases, genders and numbers. If we substitute this abbreviation by the full form of the word ("inženýra" [engineer - [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 smaller (1817). The gain of speed would be even greater would we have worked with a negative or a nonprojective variant of the parser.

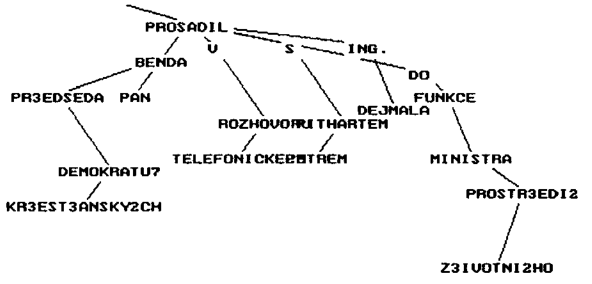
The next step is to delete further groups of words from the input sentence. Among the suitable candidates there is, for example, the prepositional phrase "v telefonickém rozhovoru" (in [the] telephone discussion). This phrase can be easily checked for grammatical correctness locally, because it has a clear left and right borders (prepositions "v"and "s"). Here we can easily solve 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 50% acceleration although it was only a cosmetic change from an abbreviation to a full word form. From the point of view of localisation 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 "inženýra" [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 following screenshots (for the sake of simplicity we demonstrate only the relevant part of derivation trees).







Let us remove the word "inženýra" 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 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 "pán" [mister] is different in Czech in case the word is "independent" and in case it is used as a title (pána vs. pana [gen.,acc.], pán vs. pan[nom.]). When we make this change we get more than 50% shorter processing time, namely 1,71s, 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 nepředpokládá spolupráci a není pravdou, že 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

"Předseda křesťanských demokratů pan Benda v telefonickém rozhovoru s Petrem Pithartem prosazoval inženýra Dejmala do funkce ministra životního prostředí."

["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.

This example and also other test data showed that the main source of ineffectivity are clauses with a big number of free modifiers and adjuncts rather than complex sentences with many clauses. These results have led us to a layered design of grammar for positive projective parsing. The core idea of this approach is the following:

Syntactic constructions which even in free word order languages may be parsed locally (certain adjectival or prepositional phrases etc.) should be parsed first in order to avoid their mutual unnecessary (from the point of view of grammar checking!) combinations. This means that the grammar should be divided into certain layers of rules (not necessarily disjunctive), which will be applied one after the other (in principle they may be applied even in cycles, but this options is not used in our implementation).

In the pivot version of our system we use the following layers:

1st layer: a metarule for processing titles and abbreviations preceding names

2nd layer: a 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 of complex sentences

5th layer: metarules for processing the left sentinel and the right hand side sentential border

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, that the results of parsing in layers provides only positive information (i.e. it 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.

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 the form of the representation of lexical 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 case: clause prep: 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 called Robust Free Order Dependency Grammar (RFODG). The limited space of this paper does not allow to present the full description of RFODG here. The definition may be found for example in [TR96].

The RFODG provides a formal base for the description of nonprojective and incorrect syntactic constructions. It introduces three measures by means of which it is possible to classify the degree of nonprojectivness and incorrectness of a particular sentence. In this paper we would like to stress one important feature of this formalism, namely the classification of the set of symbols which are used by RFODG into three types:

a) terminals and nonterminals
b) deletable and nondeletable symbols
c) positive and negative symbols

The sets under a) have the usual meaning, the sets under b) serve for the classification of syntactic inconsistencies and the sets under c) serve for their localisation. The union of terminals and nonterminals is exactly the set of all symbols used by RFODG. The same holds about the union of deletable and nondeletable symbols and also about the union of positive and negative symbols. In other words, each symbol used by RFODG belongs to exactly one set from each pair of sets under a), b) and c).

This classification therefore allows to handle rules describing both correct and erroneous syntactic constructions in a uniform way and to use a single grammar for the description of both types of syntactic constructions. Whenever a metarule describing syntactic inconsistency is used during the parsing process, a negative symbol is inserted into the tree created according to the grammar.

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 i n the input). In case that a particular rule may be applied to items A and B, a new item X is created. It is possible 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 b) and c) ("hard constraints" with an operator = may never be relaxed).

The error anticipating rules are marked by a keyword NEGATIVE at the beginning of the rule and are applied only in phases b) and c). The keyword PROJECTIVE indicates that the rule may be applied only in a projective way.

An example of a (simplified) metarule describing the attachment 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 [S97]. The grammar works with unambiguous input data (ambiguous words are represented as sets of unambiguous items). All partial parses from the first phase are used in the phases b) and c). For the purpose of testing and debugging the system we use full parsing even in the first phase.

Speeding up the performance

It is often the case that nondeterministic parsers the author of the 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 first filling the slot for the subject and then the object 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 approach makes the grammar more complicated than it is necessary and it 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. In addition, there are also some other techniques used especially in phases b) and c). The work with unambiguous input symbols allows fast parsing in the phase a) (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 error 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 prepositional 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 has 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.

Conclusion

The main purpose of the demo of our system is to demonstrate a method of grammar based grammar checking of a "free word order" language. The system is far from being ready for commercial exploitation - the main obstacle is the size of the syntactic dictionary used. Grammar based methods require a complex syntactic information about words. To build a syntactic dictionary of about 150 000 items is a task which exceeds our current capacities with respect both to manpower and funds. It would be interesting to continue the work on our system towards the development of statistical methods for this task.

References

[COL94] V.Kuboň, M.Plátek: A Grammar Based Approach to Grammar Checking of Free Word Order Languages. In: Proceedings of COLING'94, Kyoto 1994, pp. 906-910

[TR96] T.Holan, V.Kuboň, M.Plátek: Formal Tools Supporting Development of a Grammar Checker, Technical Report No.9/96, Charles University, Prague, December 1996

[CH83] J.Carbonell and P.Hayes: Recovery strategies for parsing extragrammatical language. In: American Journal of Computational Linguistics,1983 9(3-4) pp.123-146.

[K94] Z.Kirschner: CZECKER - a Maquette Grammar-Checker for Czech. In: The Prague Bulletin of Mathematical Linguistics 62, MFF UK Prague, 1994, pp. 5-30.

[M96] L.Mitjushin: An Agreement Corrector for Russian. In: Proceedings of COLING'96, Copenhagen 1996, pp. 776-781

[S97] Klaas Sikkel: Parsing Schemata - A Framework for Specification and Analysis of Parsing Algorithms, Texts in Theoretical Computer Science - An EATCS Series, ISBN 3-540-61650-0, Springer Verlag Berlin / Heidelberg / New York, 1997

---

[i] The work was supported by the following research grants: GAČR 201/96/0195, RSS/HESP No. 85/1995 and JEP PECO 2824 „Language Technologies for Slavic Languages."