AN IMPLEMENTATION OF SYNTACTIC ANALYSIS OF CZECH *

Tomáš Holan[1], Vladislav Kuboň[2], Martin Plátek[3]

[1] Department of Software and Computer Science Education
[2] Institute of Formal and Applied Linguistics
[3] Department of Theoretical Computer Science
Faculty of Mathematics and Physics
Charles University, Prague
Czech Republic
holan@ksvi.ms.mff.cuni.cz
vk@ufal.ms.mff.cuni.cz
platek@kki.ms.mff.cuni.cz

Abstract. This paper describes current results achieved during the work on parsing of a free-word-order natural language (Czech). It contains theoretical base for a new class of grammars — CFG extended for dependecies and non-projectivities — and also the description of the implementation of a parser and grammar-checker. The paper also describes some typical problems of parsing of free-word-order languages and their solutions (or discusssion of those problems), which are still subject of investigation.
The implementation described here serves currently as a testing tool for the development of a large scale grammar of Czech. Some of the quantitative data from a processing of test sentences are also included.

1 Introduction

All practically oriented linguistically based systems of natural language processing have to solve, to a certain extent, the problem of complex, linguistically adequate, correct, robust, fast and effective parsing of a particular language. Some of these requirements are contradictive — the more complex, correct, robust and linguistically adequate the system, the slower and ineffective it is.

It is quite clear that most of the interpreters of current linguistic formalisms are not effective enough to be used in a practical application, which requires "real time" processing of large quantities of texts. On the other hand, the means used in the parsing of formal languages (especially context-free languages) are effective enough, but are not prepared to deal with certain features of natural languages adequately, as for example long distance dependencies, syntactic and morphemic ambiguities etc.

During our work on the pivot implementation of a grammar checker of Czech we have tried to develop a formalism linguistically adequate and also effective enough. Our task was complicated by the nature of the language under consideration — Czech is a language with highly free word order.

* The work on this paper is a part of the joint research project PECO 2824 "Language Technologies for Slavic Languages" supported by EC and of the project "Automatic checking of grammatical errors in Czech" supported by a grant of the Grant Agency of the Czech Republic No.0475/94.
The authors are also grateful to Karel Oliva and Vladimír Petkevič for several stimuli given to us in the course of our work on the issues elaborated in the paper.

2 Non-projective Context-free Dependency Grammars

The standard CF grammars, as used for the description of formal languages, can not describe some constructions containing relations between non-neighbouring symbols. In the framework of the dependency theory these constructions are called non-projective constructions. In this chapter we introduce a class of formal grammars capable to describe these syntactic constructions.

2.1 Definition of NCFDG

In the following definition we have chosen a certain normal form of the grammar. The reason for this is the shape of the input of our parser. Two special symbols, called sentinels, are added to every original sentence, one from the left and the other from the right. Let us note that in our application we consider the categories computed by the morpholexical analysis as terminals.

Definition 1. Non-projective context-free dependency grammar (NCFDG) *is a quadruple* (N, T, S, P), *where* N, T *are sets of nonterminals and terminals,* $S \in N$ *is a starting symbol and* P *is a set of rewriting rules of the form* $A \rightarrow_L BC$ *or* $A \rightarrow_R BC, A \in N, B, C \in V$ *where* $V = N \cup T$.

The relation of immediate derivation $\Rightarrow$ *is defined as:*
$rAst \Rightarrow rBsCt$, *if* $(A \rightarrow_L BC) \in P$
$rsAt \Rightarrow rBsCt$, *if* $(A \rightarrow_R BC) \in P$,
where

$$A \in N, B, C \in V, r, s, t \in V^*$$

The relation of derivation is the transitive and reflexive closure of the relation $\Rightarrow$.

NCFD grammar G *defines language* $L(G)$ *as a set of all words* $t \in T^*$ *that can be derived from the starting symbol* S. *We say that* $L(G)$ *is recognized (generated) by* G.

Remark 2. *We can impose certain limitations on the defined language by minor changes of the definition of the relation* $\Rightarrow$. *For example, the condition* $s = EmptyString$ *reduces the relation* $\Rightarrow$ *to derivation without nonprojectivities — i.e. the same as in the standard CFGs.*

We work with fixed restrictions on the form of the rules. Variations of the type of languages are then obtained only through different types of derivation.

Definition 3. *A Tree of a word* $a_1a_2 \ldots a_n \in T^*$ *dominated by the symbol* $X \in V$, *created by NCFDG* G *is a binary branching tree* Tr *fullfilling the following conditions*

a) a node of Tr *is a triad* $[A, i, j]$, *where* $A \in V$ *and* $i, j \in 1 \ldots = n$. *The number* i *is the horizontal index of the node and* j *is the vertical index.*
b) a node $[A, i, m]$ *of* Tr *has daughters* $[B, j, p]$, $[C, k, r]$ *only if*
  1) $j < k$, $m = p + 1$, $j = i$ *and* $(A \rightarrow_L BC) \in P$ *or*
  2) $j < k$, $m = r + 1$, $k = i$ *and* $(A \rightarrow_R BC) \in P$
c) a root of Tr *is such a node of* Tr *which has no mother.*
  We can see that the root has a form $[X, i, m]$ *for some* $i, m \in 1 \ldots n$.
d) leaves of Tr *are exactly all nodes* $[a_i, i, 1], i \in 1 \ldots n$.

Remark 4. *For the sake of simplicity we are going to use in the following text the simple term Tree instead of the term Tree dominated by the symbol* S *(where* S *is a starting symbol).*

There are two differences between *Tree* and a standard parsing (derivation) tree of CFG. The first one is that a (complete) subtree of *Tree* may cover non-continuous subset of the input sentence. Second difference is that *Trees* contain enough information for building dependency trees. The basic type of the output of our parser is the dependency tree. Some constraints for our grammar-checker are introduced with the help of trees.

Definition 5. *Let Tr be a Tree of a word $a_1 a_2 \dots a_n \in T^*$ created by a NCFDG G. The* dependency tree *$Dep(Tr)$ to Tr is defined as follows: The set of nodes of $Dep(Tr)$ is the set*

$$\{[a, i]; \text{there is a leaf of } Tr \text{ of the form}[a, i, 1]\}.$$

The set of edges of $Dep(Tr)$ is the set of pairs

$$([a, i], [b, j]),$$

so that $[a, i, 1], [b, j, 1]$ are two different leaves of Tr and there is an edge $([A, i, p], [B, j, r])$ of Tr for some A, B, p, r.

Observation 6. *The language*

$$L = \{w \in (a, b, c)^*; \text{the number of the symbols } a, b, c \text{ contained in } w \text{ is equal}\}$$

may be recognized by a NCFDG G_1 and is not context-free. $G_1 = (N, T, S, P)$, $T = \{a, b, c\}$, $N = \{T, S\}$, $P = \{S \rightarrow_L aT|Ta|SS, T \rightarrow_L bc|cb\}$

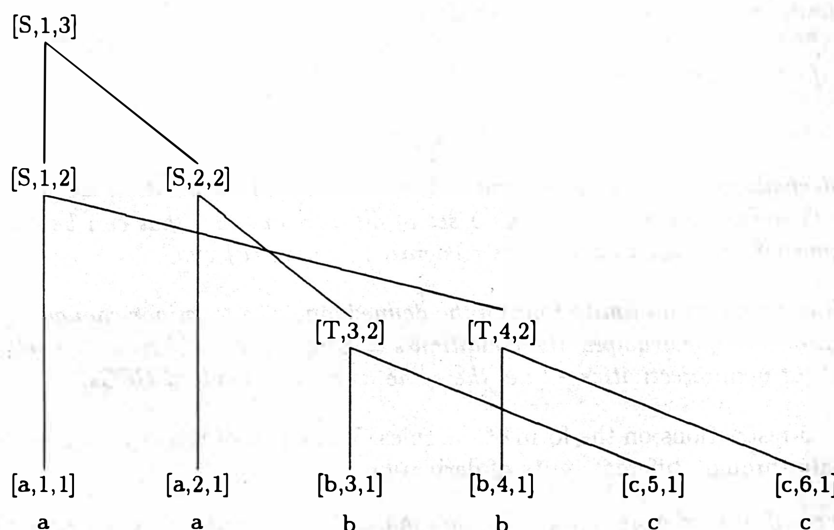

Fig. 1. A tree generated by the grammar G_1

2.2 Algorithms of recognition and parsing

From our point of view it is interesting to find out how it would be possible to compute for given words $a_1 a_2 \dots a_n$ one or all *Trees* constructed according to a given NCFDG G.

The algorithms described in the following paragraphs are based on a similar process of construction of items as for example the CYK algorithm. Those items are represented by six-tuples

$$[symbol, position, coverage, ls, rs, rule]$$

containing enough information necessary for both the reconstruction of the parsing process and for the creation of a *Trees* representing the structures of the parsed sentence.

The decision which information will be taken into account by the algorithm is then influenced by the fact whether we are trying to do only the recognition or a full parsing of a particular sentence.

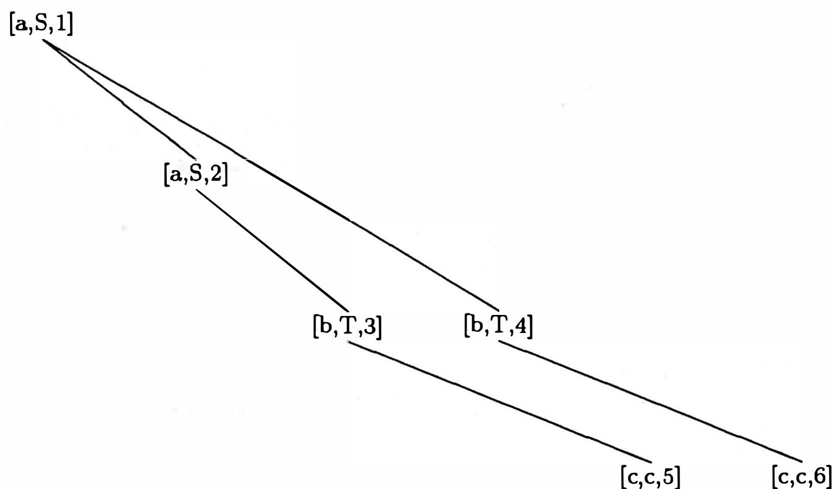

Fig. 2. Dependency tree corresponding to the tree from Fig 1.

Algorithm 7 (parsing). This algorithm works with a list D of items

$$D_i = [symbol, position, coverage, ls, rs, rule]$$

where every item corresponds to some derivations of a subsequence of the word $a_1 a_2 \ldots a_n$ according to a grammar G starting with the symbol *symbol.*

symbol represents the root symbol of the *Tree* corresponding to the derivation,

position is its horizontal index,

coverage represents the set of horizontal indices of its leaves (yield),

ls, rs are indices of items D_{ls}, D_{rs} containing the left and right daughter of the root of the particular *Tree* and

rule means the serial number of the rule according to which the item was created.

The basic idea of the algorithm is to add new items to the list as long as possible.

The list D is initialized so that for each word a_i (the word from the input sentence) we create an item

$$[a_i, i, \{i\}, 0, 0, 0]$$

Let $|D|$ be the leghth of the list D, $D[i]$ denotes the i-th item of D.

The derivation is then performed by gradual comparison of all pairs i, j, $1 <= i, j <= |D|$ and by investigating whether it would be possible to apply some rule $A \rightarrow_X BC$ ($_X$ is either L or R), which would derive the left-hand side from the right-hand side. If this is possible, a new item is created, which inherits the position either from $D[i]$ or from $D[j]$, according to whether X was L or R. The new coverage is a union of both coverages.

Two items may create a third one only if their coverages are disjoint. The difference between this algorithm and CYK is in the way how the coverage is handled by both algorithms. In CYK, the coverage is a continuous interval from-to and since the new, derived item also has to have a continuous coverage, the rule is applied only in case that j's "from" follows i's "to".

Our approach, which allows to deal with non-projective constructions, replaces this constraint by the constraint on the disjointness of both coverages. The coverage of the result is the same in both cases — the union of original individual coverages.

Let us suppose that a[1..n] is the array containing the analyzed sentence.

Step 1. (initialization)

```
for i:=1 to n do
    D[i]:=[a[i], i, {i}, 0,0,0];
NumberOfItems:=n;
```

Step 2. (derivation)

```
i:=2;
while i <= NumberOfItems do
   for j:=1 to i-1 do
     if D[i].coverage * D[j].coverage = {} then
       for rule:=1 to NumberOfRules do
       begin
         Try( D[i], D[j], rule )
         Try( D[j], D[i], rule )
       end;
```

where the procedure Try(Item1, Item2, NoOfRule) tries to apply a rule with the number NoOfRule to items Item1 and Item2.

Algorithm 8 (recognition). This algorithm is the same as Algorithm 7 with only one difference: adding of a derived item to the list D is limited only to those cases, in which D does not yet contain an item equal to the new item in the fields $[symbol, -, coverage, -, -, -, -]$.

2.3 One gap

The algorithm described above has, despite of its similarity to CYK, greater computational complexity because it is extended to non-projective constructions. We introduce a contraint which may be imposed on non-projectivity and therefore may improve the complexity of the algorithm.

We say that a relation $\Rightarrow$ is ***derivation with one gap*** if for every item *Item* derived by $\Rightarrow$ there are some

$$i, j, k, l \in 1 \ldots n, i \leq i \leq j \leq k \leq l \leq n$$

so that

$$Item.coverage = \{i \ldots j\} \cup \{k \ldots l\}$$

Unfortunately upper bound of the complexity of recognition by Algorithm 2 according to a general NCFDG is exponential. On the other hand, time and space complexity of the recognition with one gap is polynomial to the length of the sentence and to the size of the grammar, too.

In the sequel, the derivation will always mean the derivation with one gap.

3 The Treatment of Ambiguities of Word Forms

We have already mentioned one important property of natural languages — the ambiguity of word forms. It means that the same word form can belong to two or more different syntactic categories.

Our algorithm, in the same vein as CYK and other tree-based algorithms, allows to solve this problem in a strikingly simple way, changing only the initialization step of the above described algorithm from

Step 1. (initialization)

```
for i:=1 to n do
    D[i]:=[a[i], i, {i}, 0,0,0];
NumberOfItems:=n;
```

to

Step 1. (initialization)

```
NumberOfItems:=0;
for i:=1 to n do
   for j:=1 to NoOfAmbiguity[i] do
   begin
      Inc( NumberOfItems );
      D[NumberOfItems]:=[a[i,j], i,{i}, 0,0,0];
   end;
```

where a[i,j] is the j-th syntactic meaning of the word a[i].

After this first step the algorithm continues as described above, without changes, as if all input words were unambiguous.

4 Data Structures

The data structures used in our implemetation are similar to a substantially simplified form of feature structures. Our data structure is a set of attributes and their values — the value of an attribute may be only an atomic value or a list of data structures containing only attributes with atomic values. This means that the recursive use of data structures as values of attributes is limited to one recursion only.

This form of data is suitable as a intermediate step between the feature structure based formalism, in which the data are described in the main morphosyntactic lexicon of the system and the inner representation the of data.

5 Rules

The rules of the formal grammar in our system are the rules of NCFDG. If we look at the rules of our grammar from the strictly theoretical point of view, we may notice that, compared to grammar rules of a grammar of a typical programing language, there is one substantial difference between these two. The difference lies in the size of the alphabet. While the programming languages typically do not exceed the size of hunderds of symbols in the alphabet, formal description of natural languages must count with hundreds of thousands.

On the other hand lots of words of a particular natural language have very similar syntactic properties. Therefore it would be natural to use for the description of natural language grammar certain meta-rules, which will describe the syntactic properties and interactions of whole classes of words.

For example a meta-rule describing a modification of a noun by an adjective standing immediatelly on the left hand side of the noun is applicable for those word forms, where the first (left) one has a value of an attribute ***syntactic class*** — ***syntcl*** equal to ***adjective*** — ***adj*** and second (right) one has the value of the same attribute equal to ***noun*** — ***n***.

This simplified meta-rule is obviously applicable to a whole set of pairs of word-forms and therefore it may substitute all rules for every corresponding pair.

In contrast to other formalisms, the form of the description of meta-rules does not only declare constraints on the applicability of a particular rule (symbol A in case of $A \rightarrow_X BC$ rule), but also explicitly defines the order of applying the constraints together with forming the resulting data structure (symbol A) from B or from C (with respect to whether the symbol $_X$ is equal to $_L$ or to $_R$).

5.1 A formalism of meta-rules

For the description of meta-rules, we have developed a simple formalism and interpreter of this formalism. Both were designed in order to simplify the process of development of a large scale grammar of Czech for the purpose of grammar checking of Czech texts. The formalism and the

interpreter serve for the pivot implementation of the grammar checker. They can also serve as tools for testing and debugging a grammar in the process of its development.

The current version of the interpreter works with three kinds of objects:

- Input items: data structures A,B
- Temporary item: data structure P
- Output item: data structure X

Every meta-rule is described by means of a sequence of elementary instructions; some of the instructions are used in the following example:

Example:

```
;  Rule 20 - Noun in genitive
;           filling the valency frame of a preceding noun
;
   PROJECT
   IF A.SYNTCL = n THEN
      IF B.SYNTCL = n THEN
      IF B.CASE = gen THEN
                ELSE  FAIL ENDIF
               ELSE  FAIL ENDIF
      ELSE FAIL ENDIF
   ENDIF
   A.RIGHTGEN = yes
   B.CYCLE  ? yes      unfilled_frame
   P in A.FRAMESET
   P.PREP = 0
   B.CASE ? P.CASE case_disagr_in_the_frame
   X:=A
   \ P from X.FRAMESET
   IF |X.FRAMESET|=0 THEN  X.CYCLE:=yes  ELSE ENDIF
   X.rightgen := no
   OK
   END_P
```

This rule is applied to a combination of two nouns, where the second noun is in genitive. The set of constraints (starting with A.RIGHTGEN = yes and ending with IF .. ENDIF) contains two different kinds of constraints - soft constraints (operator ?) and hard constraints (operator =). In case that it is necessary to apply the constraint relaxation technique (part of an extended grammar), soft constraints are those constraints which may be relaxed. If all constraints are fulfilled, the rule causes a change of a value of the attribute RIGHTGEN in the resulting structure from "yes" to "no".

5.2 Head of a meta-rule

Any of the two parameters PROJECT and NEGATIVE may precede all commands and constraints in a particular meta-rule.

PROJECT means that the rule may be applied only in a projective way — it is in principle same as a rule of a standard CFG.

NEGATIVE means that the rule is taken into account only in an extended grammar (the explanation follows).

The handling of meta-rules is more or less procedural — they are performed from top to bottom with respect to the order of constraints and commands. A typical meta-rule of our formalism is introduced in the following example:

6 The Architecture of the Grammar Checker

Our parser is being developed for the purpose of a pivot implementation of a grammar checker for Czech. The acceptable result of the work of a grammar checker is not only the message whether the sentence of a particular language belongs to the set of well-formed sentences of that language. It is also supposed that it should find and mark the location of syntactic errors.

The location of some errors may however be determined only on the base of extralinguistic knowledge. For this reason we will talk about a grammar checker as about the program able to locate syntactic inconsistencies rather than errors. The error is usually considered as an attribute of a particular word, the syntactic inconsistency is a relation between two or more words.

Our grammar consists of two different sets of rules, a set of positive rules (without constraint relaxation) and an extended set, which contains the rules with relaxed constraints and also some special error-handling rules [6]. These rules are called negative rules. They describe typical (known) incorrect constructions and also some general rules, which are supposed to be able to handle even some unknown types of errors and therefore to make the whole system robust. The extended grammar is a proper superset of the positive grammar.

Those two sets of rules may be applied in the parser with the following three types of results:

- The sentence is recognized using positive grammar — it is correct
- The sentence was recognized using at least one negative rule — the sentence contains at least one syntactic incosistency
- The sentence was not recognized — the grammar is too weak; the system does not know anything about the sentence

In order to increase the reliability of messages about syntactic inconsistencies and also to increase the performance of the system we may apply additional constraints to the application of negative rules. For example, in our implementation we impose a constraint that *Tree* cannot contain two immediately following negative edges (this corresponds to the application of two negative rules immediately following each other).

The flow of the grammar checking is the following: The program first tries to recognize the input sentence with the positive grammar without non-projectivities. When the recognition fails, the NCFDG recognition with the positive grammar or CFG analysis with the extended grammar (projective or non-projective) may be tried.

We speak about the recognition, because the recognition is sufficient for marking the sentence as being correct or incorrect. On the other hand, if a particular sentence is not recognized with the positive grammar, it is necessary to provide full parsing with the extended grammar in order to obtain all possible syntactic inconsistencies.

Let us mention one very important remark. Even after the full parsing using the extended grammar it is not possible to provide the user with an error message. The message for the user should be created by a separate module following the parser, which will have an access to all results of parsing in case that the parser runs more than once.

At the moment our system offers four main variants of recognition or parsing:

1. recognition (parsing), positive, projective
2. recognition (parsing), positive, non-projective
3. parsing, extended, projective
4. parsing, extended, non-projective

7 Implementation

The implementation of the parser described in this section was created in 1994 and was presented to the public in February 1995 at the review meeting of the PECO 2824 JEP in Prague. It consists of app. 4000 lines of source code written in Borland Pascal.

7.1 Sample results

It seems that the derivation does not create too many new items during the processing. It means that in most derivations by rules $A \rightarrow_L BC$ or $A \rightarrow_R BC$, A is described by the same data structure as item B or C. For example, a noun modified by an adjective has usually the same syntactic data as the noun which is not modified (this holds of course only for the unambiguous structures; the morphological ambiguity of nouns is on the other hand very often solved if the noun is modified by an adjective). This results in the fact that the final number of all derivable items does not substantially exceed the number of items describing the original input sentence. The storage requirements of all items grow linearly with the length of the sentence.

For the space requirements, the number of derived items (the six-tuples from the description of the algorithms) is more important. In this section we present some examples of recognition and analysis of some sentences which may illustrate the performance of the intepreter with respect to the correctness, projectivity and length of the sentences.

Examples show number of initial items / derived items / filtered out items / Trees and memory (in bytes) / time (in seconds) requirements.

The parser was tested on IBM PC 486 DX/2 66MHz with the grammar of twenty six meta-rules.

1) Které děvčata chtěla koupit ovoce?
(Lit.: Which girls wanted [to] buy fruits?)

This sentence allows also two readings, depending whether the pronoun "které" (which) depends on "girls" or on "fruits". In the first case there is a syntactic inconsistency in the sentence (the pronoun and the subject (girls) do not agree in case), in the second case there is a long-distance dependency between the pronoun and the object (fruits). This sentence clearly illustrates the advantages of our multi-way approach: it is very difficult, if not impossible, to provide both results in a single-way mode. In the single-way mode the system usually prefers one of the possible solutions. This sentence and its two possible readings also provide an evidence for our claim that the parser has to be followed by an evaluating module, which will create the message for the user.

	Items	Space	Time
positive projective recognition	45/29/6/0	3.201 bytes	0,33s
negative projective parsing	45/467/270/1	9.080 bytes	2,15s
positive non-projective parsing	45/146/40/1	5.982 bytes	3,18s

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

(Lit.: CDP [does] not suppose cooperation with party [of] mister Sládek and it isn't true, that chairman [of] Christian democrats mister Benda enforced in telephonic discussion with Petr Pithart ing.Dejmal to function [of] minister [of] environment.)

This is a real, correct and projective sentence from newspapers — it serves as an example that the greatest problem of the current version of the interpreter are complex sentences. If no constraints are imposed on the interaction of words from different clauses in one meta-rule, the number of derived items grows dramatically. The investigations concerning the type of constraints necessary are in progress.

	Items	Space	Time
positive projective recognition	218/1998/1248/1	34.816 bytes	19,66s
positive projective parsing	218/4207/2125/16	147.284 bytes	53,06s

8 Conclusion

The main goal of this implementation of the parser was to test two basic ideas:

whether it is possible to substitute depth-first search (backtracking)[3] with the width-first search, which may, thanks to the pruning, give better results with respect to time, but usually has much bigger memory requirements, and,

whether it is possible to solve the unambiguity of word forms by creating a larger number of unambiguous items from the original ambiguous ones.

The results of the testing of our parser document that both ideas are acceptable and that much bigger role in the overall effectivity of the parsing process is played by the size of the grammar.

In the future, we are going to concetrate on the modification of our approach in order to handle efficiently complicated sentences containing many clauses, including some changes in the formalism for the description of meta rules and also in the program itself.

References

1. G. Görz : Einführung in die Künstliche Intelligenz, Addison - Wesley, 1993
2. M. A. Harrison : Introduction to Formal Language Theory, Addison - Wesley, 1978
3. J. Hric: Implementation of the Formalism for a Grammar Checker, in: Practical Application of Prolog, eds.: L. Sterling, Al Roth, 1994, Royal Society of Arts, London, UK, pp. 271-280
4. V. Kuboň, V. Petkevič, M. Plátek : Formalism for Shallow Error Checking, JRP PECO 2824, Final Research Report of the Task Adaptation and Transfer of Description Formalisms, Saarbruecken, 1993
5. V. Kuboň, M. Plátek : Robust Parsing and Grammar Checking of free Word Order Languages, in Natural Language Parsing: Methods and Formalism eds. K. Sikkel, A. Nijholt, TWLT 6, December 1993, pp. 157 - 161
6. V. Kuboň, M. Plátek : A Grammar Based Approach to Grammar Checking of Free Word Order Languages, In: Proceedings COLING 94, Vol.II, Kyoto, August, 1994, pp. 906 - 910
7. M. Plátek : The Architecture of a Grammar Checker, In: Proceedings SOFSEM '94, Milovy, 1994, pp. 85 - 90
8. N. Sikkel: Parsing Schemata, Proefschrift, Enschede,ISBN 90-9006688-8, 1993
9. T. Holan, V. Kuboň, M. Plátek: An implementation of a syntactic analysis of Czech, Technical Report No 113. of KKI MFF UK Prague
10. Z. Kirschner: CZECKER - a Maquette Grammar-Checker for Czech, The Prague Bulletin of Mathematical Linguistics 62, MFF UK Prague, 1994, pp. 5–30