# A GRAMMAR BASED APPROACH TO A GRAMMAR CHECKING OF FREE WORD ORDER LANGUAGES

Vladislav Kuboň, Martin Plátek

Faculty of Mathematics and Physics, Charles University
Malostranské nám. 25, CZ - 118 00 Praha 1, Czech Republic

## 1.INTRODUCTION

This paper shows one of the methods used for grammar checking, as it is being developed in the frame of the EC funded project LATESLAV - Language Technology for Slavic Languages (PECO 2824).

The languages under consideration in the project - Czech and Bulgarian - are both free word order languages, therefore it is not sufficient to use only simple pattern based methods for error checking. The emphasis is on grammar-based methods, which are much closer to parsing than pattern-based methods.

It is necessary to stress that we are dealing with a surface syntactic analysis. Therefore also the errors which are taken into consideration are surface syntactic errors. Our system for identification and localization of (surface) syntactic errors consists of two basic modules - the module of lexical analysis and the module of surface syntax checking. In the present paper, we will describe the second module, which is more complicated and creates the core of the whole system. Although it is not crucial for our method, we would like to point out that our approach to the problems of grammar checking is based on dependency syntax.

Let us illustrate the degree of freedom of the word order, which is provided by Czech, one of the languages under consideration in the project. If we take a sentence like
"Označený (Adj. masc., Nom/Gen Sg.) soubor (N masc., Nom/Gen Sg.) se (Pron.) nepodařilo (V neutr., 3rd pers. Sg) úspěšně (Adv.) otevřít (V inf.)" (The marked file failed to be opened sucessfully); word-for-word translation into English "Marked file itself failed succesfully to open", we may modify the word order for instance in the following way:

Nepodařilo se úspěšně otevřít označený soubor.
Označený soubor se úspěšně otevřít nepodařilo.
Označený soubor se nepodařilo otevřít úspěšně.
Úspěšně otevřít označený soubor se nepodařilo.
Nepodařilo se označený soubor úspěšně otevřít.
Nepodařilo se označený soubor otevřít úspěšně.
etc.

The only thing concerning the word order, which we can guarantee, is that the above introduced list of syntactically correct sentences does not exhaust all possibilities, which are given by the combinations of those six Czech words[1]. The example also shows, that although the word order is very free, there are certain limitations to that freedom, as e.g. the adjective - noun group ("označený soubor"), which is closely bound together in all of the sample sentences, but may not be bound together in some other context - cf. "...soubor Karlem včera večer označený jako špatný...." (...file [by Karel] yesterday evening marked as wrong...).

The approach which we have chosen for the development of the grammar checker for free word order languages is based on the idea of reducing the complicated syntactic structure of the sentence into a more simple structure by means of deleting those words from the input sentence which do not cause any error.

Let us take as an example the (ungrammatical) English sentence quoted in [3]: *"*The little boys I mentioned runs very quickly."* The error may be extracted by a stepwise deletion of the correct parts which do not affect the (non)correctness of the sentence. We will get successively the sentences *"*The boys I mentioned runs very quickly"*, *"*boys I mentioned runs very quickly"*, *"*boys runs very quickly"*, *"*boys runs quickly"*, *"*boys runs"*.

[1]As mentioned above, we are concerned with surface syntactic shape of the Czech sentences and thus we leave aside the semantic relevance of the word order variations due to their different topic - focus articulation. For a detailed discussion of these phenomena, see esp. [1] and [7].

The example shows that it is useful to use a model which is able to deal with deletions in a natural way. We use the nondeterministic list automata (NLA); a list automaton works with a (doubly linked) list of *items* rather than with a tape. The version of the NLA which is used in our project is described briefly in the following sections.

## 2.ERROR CHECKING AUTOMATON

The core module of our system is the Error Checking Module. It recognizes grammatical correctness of a given portion of text, or, in other words, it distinguishes the grammatically correct and grammatically incorrect subsequences (not necessarily continuous) of lexical elements in the input text.

The input of the Error Checking Module (ECM) consists of the results of the morphological and lexical analysis. The exact form of the input elements is thoroughly described in [5]. For the purpose of this paper it is only necessary to say that the data, representing one lexical element, are contained in one complex feature structure. The attributes are divided into two groups, input and output attributes. The ECM tries to reduce the input sequence by means of deleting some symbols. The deleted symbols are stored. They create the history of simplifications of the input text.

The whole process is nondeterministic - if there are more variants how to delete the symbols, all of them are sooner or later taken into account.

For the purpose of the grammar checker, we use a slightly modified version of NLA, called Error Checking Automaton (ECA). ECA has a two-level storage, with a two-way linear list on each level composed of data items (fields, cells). In the list there are two distinguished items: one at the leftmost end and the other at the rightmost end of the list. These items are called sentinels.

The first level represents the input and the working storage of ECA, the other one ECA's output. ECA has a finite state control unit with one head moving on a linear (doubly linked) list of items. In each moment the head is connected with one particular cell of the list ("the head looks at the visited field"). The actions of the working head are delimited by the following four types of basic operations which the head may perform on the list: MOVE, DELETE, INSERT, and RESTART. The operations of the type MOVE ensure the bi-directional motion of the head on the list. The DELETE operations delete the input field in the input level. The INSERT operations add a field with a symbol to the output level, more exactly: to the close neighborhood of the visited field. The RESTART operation transfers ECA from the current configuration to the (re)start configuration, i.e. to the configuration in which ECA is in the initial (unique) state.

The processing of the ECA is controlled by rules, written in a formalism called DABG (Deletion Automata Based Grammar), which was developed especially for the project LATESLAV. It is described in detail in [5]. The theoretical background for erasing automata of the above described kind can be found in [6] and [2].

The ECM is logically composed of the following three components:

(a) list automaton **P** of the type ECA;
(b) list automaton **N** of the type ECA;
(c) control module **C**.

### 2.1. The automaton P

The automaton works in cycles between (re)starts. The function of the automaton **P** is to apply one rule of the control grammar to the input during one cycle. That means to decide nondeterministically which finite subsequence of the text in the input level of the list is correct according to one rule of the control grammar, and to delete this part from the input level. After that it continues the work in the next cycle.

This means that if the input text is error free, the automaton **P** gradually repeats cycles and deletes (in at least one branch of its tree of computations) all the input elements (except for the sentinels).

The automaton **P** accepts the input sequence, if the computation of **P** finishes by deleting all the items (except for the sentinels) from the input level of the list.

Notation:

**L(P)** is a set of strings accepted by **P**.

**rs(P,w)** = {$w_1$, where $w_1$ is a word, which is a result of one cycle performed on the word w by the automaton **P**}

We can see that the following two facts hold, due to the restarting nature of computations of **P**:

**Fact 1**: Let w be a word from L(P), then $rs(P,w) \cap L(P) \neq \emptyset$.

**Fact 2**: If w is not a word from L(P), then $rs(P,w) \cap L(P) = \emptyset$.

Two basic principles how to formulate rules for the automaton **P** for a given natural language L follow from the above mentioned facts:

1) **P** contains only those (deleting) rules, for which there exists a sentence in L which will remain syntactically correct after the application of the rule.

2) There may not be a syntactically incorrect sequence of words from L which would be changed to a syntactically correct sentence of L by means of the application of a rule from **P**.

**Strong rules** (S-rules) are the rules which meet the following requirement:

3) Any application of a rule will keep both correctness and incorrectness of the input sequence.

Clearly the S-rules meet also the requirements 1) and 2).

The subautomaton of **P**, which uses S-rules only, is called $\mathbf{P_S}$.

One cycle (one compound step) of an automaton **P** (or $\mathbf{P_S}$) can be described from the technical point of view as follows:

First, the automaton searches through the input level for the place where there is a possibility to apply one of the deleting rules to the input level of the automaton. In the positive case the deleting rule is applied and **P** (or $\mathbf{P_S}$) returns to the initial configuration (restarts).

### 2.2. The automaton N

The task of the automaton **N** is to find in the input text a minimal limited error, to make a correction of it (cf. the following definition). In one compound step the automaton **N** performs (nondeterministically) the following actions:

First, similarly as the automaton **P**, **N** locates the place where to apply a rule of its control grammar to the input level. Then it checks whether there is an error in the close neighborhood of that place. In the positive case it marks the occurrence of the error at the output level of the list and corrects the input level of the list by deleting some items from the environment of the current position of the head.

**Definition**: The **limited error** is a string **z**, for which there are no **y, w** such that the string **yzw** is a (grammatically) correct sentence (of a given language L). If **z** can be written in the form of

$$\mathbf{z = v_0 u_1 v_1 u_2 v_2 \cdots u_k v_k}$$

and there are also strings **s,r** such that $\mathbf{su_1u_2 \ldots u_kr}$ is a grammatically correct sentence, $\mathbf{u = u_1u_2 \ldots u_k}$ is called a correction of **z**.

A **minimal limited error** is a string **z**, which is a limited error and there is no possibility how to shorten **z** from either side preserving the property of being a limited error for **z**.

### 2.3. The control module C

The **C** module is a control submodule of the entire ECM module. At the beginning of the operation of ECM, the **C** module calls the automaton **P**, which works as long as it is possible to shorten the input level list by deleting its items. As soon as the automaton **P** cannot shorten the list in the input level any more and the input level does not contain only the sentinels, **C** calls the module **N**. This automaton removes one error from the input level, makes an error mark and transfers the control back to the **C** module, which invokes the automaton **P** again. Thus, the submodule **C** repeatedly invokes the automata **P** and **N** (switching between them) as long as they are able to shorten the sequence of input elements. If there are more possibilities at a certain moment of computation, the automaton **P** chooses only one of them, **C** stores the other into the stack and it tries to apply another rule to the modified input. That means that **C** is searching the tree of possible computations depth-first.

In any stage of selection of rules for **P** and **N** there may be some input sentences, which contain either syntactically correct subsequences of words which cannot be handled by the rules of **P**, or syntactic errors which are not covered by the rules of **N**. In this case both automata stop and the input level still contains a subsequence of input elements. Its contingent final emptying is ensured by the **C** module, which marks this fact at the output level. Then the **C** module transfers control to the next phase of the whole system.

At this point it is necessary to clarify, what kind of output structure is built by ECM. As we have already mentioned, our approach to the problem is oriented towards dependency syntax[2].

All the rules of control grammar for **P** and **N** delete the depending word from the input and put it into the output attribute of the governing word. At the end of the process there is a tree, which contains the information about all the words from the input, about the order of deletions and also all error marks made by the automaton **N**.

The switching between **P**, **N** and **C** guarantees that any possible path in the tree of computation will result in a complete structure of a deletion tree.

The current best deletion tree is then compared to any new deletion tree. If the new tree is better (e.g. it contains a smaller number of errors or contains errors with a smaller weight), it is then stored for further use as the new current best result.

At the end of the whole process we have the "best" result (or a set of best results, e.g. when there are more possibilities how to parse the sentence), which contains all relevant information about errors present in the input sentence.

At the current stage of the work we have decided to distinguish as considerable only the following two types of errors:

a) Only one call of **N** was used and the whole process of deletions is completed by **P** and **N** only.

b) If there were only the rules of $\mathbf{P_S}$ and **N** applied to a particular path in a tree of computation.

Clearly the tree with error marks of the type a) will be among the best results of the **C** for any reasonable comparison of results. We have to assign a slightly smaller weight to the errors of the type b).

## 3. CONCLUSION

In the previous paragraphs we have sketched the specifications for a grammar based grammar checker for a free word order language, as it is viewed from the perspective of our approach to the project LATESLAV. The main goal of the project is to develop a methodology of solving the problem of "grammar based" grammar checking of relevant languages. We hope that the ideas presented in the paper may help us to achieve this goal successfully.

The paper shows that it is possible to make a clear distinction between two parts of the grammar checker, namely the part (automaton P) which is based on rules which describe correct subparts of a given input sentence and therefore is very close to a standard dependency syntactic parsing, and between the part (automaton N) which is based on rules which are to a great extent similar to the rules used in standard "pattern based" approach to grammar checking. The combination of these two parts provides a tool more powerful (in the formal sense) than if both approaches are applied in isolation.

The architecture of the system also makes it possible to re-use the existing linguistic knowledge about the computational grammar of Czech. It also allows to divide the enormous task of implementation of a grammar checker in smaller specialized subparts (e.g. rules for **P** and **N**), which may be developed independently.

As shown in [4], similar specifications, based on the same principle, can also be used in the area of robust syntactic parsing.

[2]However, the use of DABG for creating the control grammars for **P** and **N** is not limited to dependency based approach only. Both the data structures (feature-structure based) and the DABG formalism allow to formulate rules which create the constituent structure of the sentence at the output level.

## References

[1] Hajičová E.: "Free" word order described without unnecessary complexity. Theoretical Linguistics 17, 1991, pp. 99-106.
[2] Jančar P., Mráz F., Plátek M.: A taxonomy of forgetting automata. In: Proceedings of MFCS'93, Gdansk, Poland, August 1993, LNCS 711, Springer 1993, pp. 527-536.
[3] Jančar P., Mráz F., Plátek M.,Vogel J.: Deleting automata with a restart operation, submitted for MFCS'94.
[4] Kuboň V., Plátek M.: Robust parsing and grammar checking of free word order languages, In: Proceedings of the 6th Twente Workshop on Language Technology, Universiteit Twente, Enschede, December 1993, pp. 157-161.
[5] Kuboň V., Petkevič V., Plátek M.: Formalism for shallow error checking; JRP PECO 2824, In: Final Research Report of the Task: Adaptation and Transfer of Description Formalisms, Saarbruecken, 1993
[6] Oliva K., Plátek M., Seznamové automaty a typy popisu povrchové syntaxe (List Automata and the Types of Description of Surface Syntax). In: SOFSEM'90, Janské Lázně 1990, pp.61-64, Vol.II.
[7] Sgall P., Hajičová E. and Panevová J.: The meaning of the sentence in the semantic and pragmatic aspect, Reidel: Dordrecht and Academia: Prague, 1986.

The work described in this paper was performed in the frame of the Joint Research Project PECO 2824.