New Meta-grammar Constructs in Czech Language Parser synt*

Aleš Horák and Vladimír Kadlec

Faculty of Informatics, Masaryk University Brno
Botanická 68a, 602 00 Brno, Czech Republic
{hales, xkadlec}@fi.muni.cz

Abstract. In this paper, we present and summarize the latest development of the Czech sentence parsing system **synt**. The presented system uses the meta-grammar formalism, which enables to define the grammar with a maintainable number of meta-rules. At the same time, these meta-rules are translated into rules for efficient and fast head driven chart parsing supplemented with evaluation of additional contextual constraints. The paper includes a comprehensive description of the meta-grammar constructs as well as actual running times of the system tested on corpus data.

1 Introduction

The development of a grammar for syntactic analysis of natural language texts is a tedious process which always tends to keep adding new rules for capturing uncovered language phenomena up to the moment, where the number of rules becomes hardly maintainable. Such "rule overload" is solved with several competing approaches ranging from stochastic parsing [1], through various automatic learning algorithms [2] to loosening the strictness of the analysis with a shallow parsing algorithm [3].

In the synt system approach, a deep syntactic representation of sentence is required, since the resulting analysis is used as an input for the Normal translation algorithm [4] producing a meaning representation of NL expressions with constructions of Transparent intensional logic. Therefore, we have decided to apply the meta-grammar concept with three consecutive grammar forms. The underlying analyzer is an augmented head driven parser based on a context-free backbone with interleaved evaluation of contextual constraints.

2 Description of the System

The meta-grammar concept in synt consists of three grammar forms denoted as G1, G2 and G3. Human experts work with the meta-grammar form, which

* This work has been partly supported by Czech Science Foundation under the project 201/05/2781 and by Grant Agency of the Academy of Sciences of CR under the project 1ET400300414.

encompasses high-level generative constructs reflecting the meta-level natural language phenomena like the word order constraints, and enable to describe the language with a maintainable number of rules. The meta-grammar serves as a base for the second grammar form which comes into existence by expanding the constructs. This grammar consists of context-free rules equipped with feature agreement tests and other contextual actions. The last phase of grammar induction lies in the transformation of the tests into standard rules of the expanded grammar with the actions remaining to guarantee the contextual requirements.

Meta-grammar (G1) The meta-grammar consists of global order constraints that safeguard the succession of given terminals, special flags that impose particular restrictions to given non-terminals and terminals on the right hand side (RHS) and of constructs used to generate combinations of rule elements. Some of these meta-grammar constructs have already been described in [5], in this paper we present a summary of all the constructs including the latest additions.

We use the arrow in the rule for specification of the *rule type* (`->`, `-->`, `==>` or `===>`). A little hint to the arrow form meaning can be expressed by 'the thicker and longer the arrow the more (complex) actions are to be done in the rule translation'. The smallest arrow (`->`) denotes an ordinary CFG transcription and the thick extra-long arrow (`===>`) inserts possible inter-segments between the RHS constituents, checks the correct order of enclitics and supplies several forms of the rule to make the verb phrase into a full sentence.

The *global constructs* (`%enclitic`, `%order` and `%merge_actions`) represent universal simple regulators, which are used to inhibit some combinations of terminals in rules, or which specify the actions that need some special treatment in the meta-grammar form translation.

The main *combining constructs* in the meta-grammar are `order()`, `rhs()` and `first()`, which are used for generating variants of assortments of given terminals and non-terminals.

```
/* budu se ptat - I will ask */
clause ===> order(VBU,R,VRI)
/* ktery ... - which ... */
relclause ===> first(relprongr) rhs(clause)
```

The `order()` construct generates all possible permutations of its components. The `first()` and `rhs()` constructs are employed to implant content of all the right hand sides of specified non-terminal to the rule. The `rhs(N)` construct generates the possible rewritings of the non-terminal `N`. The resulting terms are then subject to standard constraints, enclitic checking and inter-segment insertion. In some cases, one needs to force a certain constituent to be the first non-terminal on the RHS. The construct `first(N)` ensures that `N` is firmly tied to the beginning and can neither be preceded by an inter-segment nor any other construct.

There are several generative constructs for defining rule templates to simplify the creation and maintenance of the grammar. One group of such constructs is

formed by a set of `%list_*` expressions, which automatically produce new rules for a list of the given non-terminals either simply concatenated or separated by comma and co-ordinative conjunctions.

A significant portion of the grammar is made up by the verb group rules (about 40 %). Therefore we have been seeking for an instrument that would catch frequent repetitive constructions in verb groups. The obtained addition is the `%group` keyword illustrated by the following example:

```
%group verbP={
  V:    verb_rule_schema($@,"(#1)")
        groupflag($1,"head"),
  VR R: verb_rule_schema($@,"(#1 #2)")
        groupflag($1,"head"),
}

/* ctu/ptam se - I am reading/I am asking */
clause ====> order(group(verbP), vi_list)
  verb_rule_schema($@,"#2")
  depends(getgroupflag($1,"head"), $2)
```

Here, the group `verbP` denotes two sets of non-terminals with the corresponding actions that are then substituted for the expression `group(verbP)` on the RHS of the `clause` non-terminal. In order to be able to refer to verb group members in the rules, where the group is used, any group term can be assigned a *flag* (any string). By that flag an outside action can refer to the term later with `getgroupflag` construct.
Many rules, e.g. those prescribing the structure of a `clause`, share the same rule template — they have the same requirements for inter-segments filling and the enclitics order checking as well as the RHS term combinations. To enable a global specification of such majority of rules, we provide a *rule template* mechanism, which defines a pattern of each such rule (the rule type and the RHS encapsulation with some generative construct).

Some grammatical phenomena occur very rarely in common texts. The best way to capture this sparseness is to train rule probabilities on a large data bank of derivation trees acquired from corpus sentences. Since preparation of such corpus of adequate size (at least tens of thousands of sentences) is a very expensive and tedious process, we have for now overcome this difficulty with defining *rule levels*. Every rule without level indication is of level 0. The higher the level, the less frequent the appropriate grammatical phenomenon is, according to the guidance of the grammarian. Rules of higher levels can be set on or off according to the chosen level of the whole grammar.

Apart from the common generative constructs, the meta-grammar comprises feature tagging actions that specify certain local aspects of the denoted (non-)terminal. One of these actions is the specification of the head-dependent relations in the rule — the `head()` and `depends` construct which allow to express the dependency links between rule terms.

The Second Grammar Form (G2) As we have mentioned earlier, several predefined grammatical tests and procedures are used in the description of context actions associated with each grammatical rule of the system. The pruning actions include:

- grammatical case test for particular words and noun groups
- agreement test of case in prepositional construction
- agreement test of number and gender for relative pronouns
- agreement test of case, number and gender for noun groups
- type checking of logical constructions

```
np -> adj_group np
  rule_schema($@, "lwtx(awtx(#1) and awtx(#2))")
  rule_schema($@, "lwtx([[awt(#1),#2],x])")
```

The contextual actions `propagate_all` and `agree_*_and_propagate` propagate all relevant grammatical information from the selected non-terminals on the RHS to the one on the left hand side of the rule.

The `rule_schema` action presents a prescription for building a logical construction out of the sub-constructions from the RHS. Each time a type checking mechanism is applied and only the type-correct combinations are passed through.

Expanded Grammar Form (G3) The feature agreement tests can be transformed into context-free rules. For instance in Czech, similar to other Slavic languages, we have 7 grammatical cases (nominative, genitive, dative, accusative, vocative, locative and instrumental), two numbers (singular and plural) and three genders (masculine, feminine and neuter), in which masculine exists in two forms — animate and inanimate. Thus, e.g., we get 56 possible variants for a full agreement between two constituents.

The number of rules naturally grows in the direction G1 < G2 < G3. The current numbers of rules in the three grammar forms are 253 in G1, 3091 in G2 and 11530 in G3, but the grammar is still being developed and enhanced.

3 Parser

We restrict our work to lexicalized grammars, where terminals can only appear in lexical rules in the form of $A \rightarrow w_i$. This restriction allows us to simplify the implementation and it also enables to separate a lexicon from the grammar. The parsing module of `synt`, the `libkp` library, provides an efficient implementation of standard parser tasks:

- syntactic analysis of sentences in natural language based on context-free grammars that can be large and highly ambiguous;
- efficient representation of derivation trees;
- pruning of the trees based on the application of contextual constraints;
- selecting n most probable trees based on computed probabilities of edge values (e.g. the frequency characteristics obtained from tree-banks);

Princeton WordNet: awaken:1, wake:5, waken:1, rouse:4, wake up:1
Definition: cause to become awake or conscious
FIMU Vallex: budit:1 impf. / vzbudit:1 pf. / probudit:1 pf.
frame: CAUSE <cause:4>$^{obl}_{co1}$ VERB PAT <person:1>$^{obl}_{koho4}$
example: probudil mě hluk pf. / that noise awoke me
example: budí mě budík impf./ an alarm clock wakes me up

Fig. 1. An example of FIMU Vallex verb frame.

The parsing process consists of several steps. Firstly, the packed shared forest [6] is produced by the standard CF parsing algorithm. Then the contextual constraints are applied. Finally, most probable trees are selected. The order of the last two steps can be reversed. We use this multi-pass approach, because all these functions are implemented as plug-ins that can be modified as needed or even substituted with other implementations. For example, we have compared four different parsing algorithms which use identical internal data structures (Earley's top-down and bottom-up chart parser [6], head-driven chart parser [7] and Tomita's GLR [8]. All these implementations produce the same structures, thus applying contextual constraints or selecting n best trees can be shared among them.

3.1 Evaluation of Contextual Constraints

The contextual constraints (or actions) defined in the meta-grammar can be divided into four groups:

1. rule-tied actions
2. agreement fulfillment constraints
3. post-processing actions
4. actions based on derivation tree

The rule-based probability estimations are solved on the first level by the rule-tied actions, which also serve as rule parameterization modifiers.

Agreement fulfillment constraints serve as chart pruning actions and they are used in generating the expanded grammar G3. The agreement fulfillment constraints represent the functional constraints, whose processing can be interleaved with that of phrasal constraints.

The post-processing actions are not triggered until the chart is already completed. Actions on this level are used mainly for computation of analysis probabilities for a particular input sentence and particular analysis. Some such computations (e.g. verb valency probability) demand exponential resources for computation over the whole chart structure. This problem is solved by splitting the calculation process into the pruning part (run on the level of post-processing actions) and the reordering part, that is postponed until the actions based on derivation tree.

The actions that do not need to work with the whole chart structure are run after the best or n most probable derivation trees are selected. These actions are used, for example, for determination of possible verb valencies within the input sentence, which can produce a new ordering of the selected trees, or for the logical analysis of the sentence.

The edge probability computations use information from several supporting resources, such as a tree-bank of testing sentences or a list of verb valencies. The latter one is currently the most promising resource, since we are working on an enhanced list of verb frames (FIMU Vallex), see [9], featuring syntactic dependencies of sentence constituents, their semantic roles and links to the corresponding Czech WordNet classes. An example of such verb frame is presented in the Figure 1. The list currently contains more than 3000 verbs which, when gathered in synonymic groups, share about 1700 verb frames.

3.2 Implementation

In the `libkp` library, every grammar rule has zero, one or more semantic actions. The actions are computed bottom-up serving the purpose of:

- computing a value used by another action on the higher level;
- throwing out incorrect derivation trees.

For example, the following grammar rule for genitive constructions in Czech has three semantic actions:

```
npnl -> np np +0.0784671532846715
  test_gen ( $$ $2 )
  prop_all ( $$ $1 )
  depends:1 ( $$ $1 $2 )
```

First line contains a grammar rule with its frequency obtained from a tree-bank. The contextual constraints are listed on the lines bellow it. The number 1 after the colon represents an internal classification of the action. We can turn an evaluation of actions with specified type on and off. The `$$` parameter represents the return value. The `$`k parameter is a variable where we store a value of k-th nonterminal of the rule. Notice that the presented notation is not entered directly by users. It is generated automatically from the meta-grammar G1.

The representation of values It was shown that parsing is in general NP-complete if grammars are allowed to have agreement features [10]. The pruning constraints in `libkp` are weaker than general feature structures. A node in the derivation tree has only limited number of values, e.g. the number of values for noun groups in our system is at most 56.

During the run of the chart based parsing algorithm the results of the parsing process are stored in a packed shared forest of Earley's items [6]. To compute the values, we build a new forest of values instead of pruning original packed shared forest. The worst-case time complexity for one node in the forest of values

Fig. 2. Example of the forest of values.

is therefore 56^{δ}, where δ is the length of the longest right-hand side grammar rule. Notice that this complexity is independent on the number of words in input sentence.

The values in the forest of values are linked with Earley's items. An item contains a single linked list of its values. Each value holds a single linked list of its children. The child is one dimensional array of values. This array represents one combination of values that leads to the parent value. Notice that there can be more combinations of values that leads to the same value. The i-th cell of the array contains a reference to a value from i-th symbol on the RHS of the corresponding grammar rule. The i-th symbol has not to be used to compute the parent value. We use only reference to Earley's item from such unused cell.

The Figure 2 shows an example representing rule `npnl -> np np` and containing three Earley's items. Each item on the RHS of the rule contains two values, *value1* and *value2*. This gives us four possible combinations. The semantic action computes from combinations *value1* $\times$ *value2* and *value2* $\times$ *value1* the same value *value4*. The combination *value2* $\times$ *value2* was classified as incorrect (by the action), so it is not here.

Table 1. Results of running the `synt` parser on 10.000 corpus sentences (run on Intel Xeon 2.2GHz).

#sentences	10000 sentences from DESAM corpus
#words	191034 words
Maximum sentence length	155 words
Minimum sentence length	2 words
Average sentences length	19.1 words
Time of CFG parsing	5 minutes 52 seconds
Time of evaluating constraints (actions)	38 minutes 32 seconds
Overall time with freeing memory	46 minutes 9 seconds
Average #words per second	68.97
Size of the log file	1.2 GiB
#accepted sentences	9208

3.3 Results

In the current stage of the meta-grammar development, `synt` has achieved an average of 92.08 % coverage on corpus sentences with about 84 % cases where the correct syntactic tree was present in the result. However, the process of determining the correct tree is still premature.

An average time of analysis of one sentence from the corpus data was 0.28 seconds.[1] More details of timing results can be found in the Table 1.

4 Conclusions

The presented parsing system `synt` has already proven its abilities in analysis of running texts in both speed and coverage of various sentence types. With continuous development of the grammar in order to obtain coverage of the necessary language phenomena, we obtain a quality general purpose system for deep syntactic analysis of natural language texts even for language with such extent of non-analytical features as the Czech language is.

The current development of the system lies mainly in probabilistic ordering of the obtained analyzes with the usage of language specific features such as augmented valency frames in FIMU Vallex.

References

1. Rens Bod. An efficient implementation of a new dop model. In *EACL 2003*, pages 19–26, 2003.
2. Michael Collins and Nigel Duffy. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL 2002*, pages 263–270, 2002.
3. A. van den Bosch and S. Buchholz. Shallow parsing on the basis of words only: A case study. In *ACL 2002*, pages 433–440, 2002.
4. Aleš Horák. *The Normal Translation Algorithm in Transparent Intensional Logic for Czech.* PhD thesis, Faculty of Informatics, Masaryk University, Brno, 2002.
5. P. Smrž and A. Horák. Large scale parsing of Czech. In *Proceedings of Efficiency in Large-Scale Parsing Systems Workshop, COLING'2000*, pages 43–50, Saarbrucken: Universitaet des Saarlandes, 2000.
6. J. Earley. An efficient context-free parsing algorithm. In *Communications of the ACM*, volume 13, pages 94–102, 1970.
7. M. Kay. Algorithm schemata and data structures in syntactic processing. In *Report CSL-80-12*, Palo Alto, California, 1989. Xerox PARC.
8. M. Tomita. *Efficient Parsing for Natural Languages: A Fast Algorithm for Practical Systems.* Kluwer Academic Publishers, Boston, MA, 1986.
9. Aleš Horák and Dana Hlaváčková. Transformation of wordnet czech valency frames into augmented vallex-1.0 format. In *Proceedings of LTC 2005*, Poznan, Poland, 2005. accepted for publication.
10. G. E. Barton, R. C. Berwick, and E. S. Ristad. *Computational complexity and natural language.* MIT Press, Cambridge, Massachusetts, 1987.

[1] The average running time includes generation of log file messages.