UDC 519.812.4

J. ANTIDZE[1], N. GULUA[2]

SOFTWARE TOOLS FOR COMPUTER REALIZATION OF MORPHOLOGICAL AND SYNTACTIC MODELS OF GEORGIAN TEXTS

[1]Tbilisi State University, Vekua Scientific Institute of Applied Mathematics, Tbilisi, Georgia,
Phone : 995 32 545079 Mobile : 855231237,
E-mail: jeantidze@yahoo.com ,

[2]Sukhumi State University, |Tbilisi, Georgia,
E-mail: ngulua7@mail.ru

Abstract. Developed software tools is a software system designed for syntactic and morphological modeling of natural language texts. The tools are efficient for a language, which has free order of words and developed morphological structure like Georgian. For instance, a Georgian verb has several thousand verb-forms. It is very difficult to express morphological analysis' rules by finite automaton and it will be inefficient as well. Resolution of some problems of complete morphological analysis of Georgian words is impossible by finite automaton. Splitting of some Georgian verb-forms into morphemes requires non-deterministic search algorithm, which needs many backtrackings. To minimize backtrackings, it is necessary to put constraints, which exist among morphemes and verify them as soon as possible to avoid false directions of search. It is possible to minimize backtracking and use parameterized macros by our tools. Software tool for syntactic analysis has means to reduce rules, which have the same members in different order. Thus, proposed software tools have many means to construct efficient parser, test and correct it. We realized morphological and syntactic analysis of Georgian texts by these tools. In presented article, we describe the software tools and its application for Georgian language.
Keywords: Software tools, Parsing algorithm, Backtracking, Syntactic analyzer, Constraints.

INTRODUCTION

The computer morphological analysis of Georgian words is one of the main components for solving such problems as machine translation from Georgian language to the other languages, as well as the automated checking of orthography of Georgian texts, and some problems of artificial intelligence, which require computer processing of Georgian texts. The complete system for computer morphological analysis of Georgian words does not exist yet. If we need to use Georgian language to communicate with computer, the solving of mentioned above problem is very urgent.

For solving this problem using of finite automaton, which is widely used for the languages from Western Europe, is not feasible. This is happening because of some verb-forms of Georgian language require backtracking, which is impossible with finite automaton. From the other side, using of full search algorithm slows the process of morphological analysis. For this reason, we formed a method, which is making the analysis process faster, compare to full search algorithm ([1]). This method uses constraints to establish correct morpheme's selection. Already separated presumable morphemes from word, morphological analysis tool checks it on satisfaction of their constraints. If the constraint is satisfied, the tool continues separation of other morphemes in opposite case it performs backtracking to search the new alternatives and rejects the last separated morpheme. In this way, the process of removing of incorrect alternatives happens in advance, what speeds up the searching process. The constraints are logical expressions, which we can compose from the features of morphemes. The tool checks, if separated morpheme's feature has particular value, which defines correctness of the separation. We compose the values of morphemes' features according to morphology of Georgian language.

Under complete computer morphological analysis, we understand all valid splitting of a word-form in morphemes and establishment of morphological categories for each splitting. The definition contains ambiguities of words. The following ambiguity is widespread:

1. Graphical coincidence of different verb-forms (by meaning) in presence circle, which have the same root. For instance, verb-form "agebs", which may mean loss (many) or build (plan) and so on;

2. Graphical coincidence of a verb-form with its infinitive, for instance, "amoxsna" may mean, "resolution" or "he has resolved";

3. In time of splitting of verb-form, graphical coincidence of morphemes from different neighboring classes, for instance, "a" as the preverbal or vowel prefix or first letter of a verb's root in the following verb-forms: "a-a-alebs", "a-alebs" and "aldeba". When we see first letter of the verb-form "aaalebs", we cannot say, which morpheme we have, before we have seen following two letters. In first example, first "a" is preverbal. In second example, first "a" is vowel prefix and in third example first "a" is first letter of the root "al". This means, that Georgian verbs splitting in morphemes needs at least parsing algorithm for LL(2) grammar ([2]), i.e. complete morphological analysis of Georgian words by finite automaton is impossible.

In the second case, morphological analysis for verb-form "amoxsna" must give two different parsing: one- for infinitive and second - for verb-form. For this, we need nondeterministic algorithm. Deterministic algorithm cannot give two different parses for the same word-form. Thus, deterministic algorithm is not valid for complete morphological analysis of Georgian words. All author fulfilled morphological analyses for Georgian words by finite automaton or by deterministic algorithm ([3, 4]). For complete morphological analysis, we must apply non-deterministic algorithm, for instance, from left to right in depth search algorithm with backtrackings. As far as backtrackings take down the speed of the algorithm, we must find a method, which reduces them. Such possibility exists. We can exclude morphemes, which conflict with found morphemes at a moment. In other case, we can divide morphemes in classes so, that one representative of each class will meet as maximum one times in a word-form. Among morphemes of a verb-form are important roots. We can divide roots into classes so, that each morpheme, which can meet in a word-form, will indicate definitely a morphological category. All this reduces backtrackings and establishing morphological categories considerably. After this, the establishment of morphological categories of a word is easy. We realized complete morphological analysis of Georgian words by the tools ([5-7]).

SOFTWARE TOOLS

The "Software Tools for Morphological and Syntactic Analysis of natural Language Texts" is a software system designed for the processing of natural language texts. We use the system to analyze syntactic and morphological structure of the natural language texts, using specific formalism, which we created for this purpose, allow us to write down syntactic and morphological rules defined by particular natural language grammar. This formalism represents the new, complex approach, which solves problems of morphological and syntactic analysis for some natural language. We implemented a software system according to this formalism ([1]). One can realize syntactic analysis of sentences and morphological analysis of word-forms with this software system. We designed several special algorithms for this system. Using the formalism, which is described in ([8, 9]), is very difficult to use for Georgian language, as far as expressing of some morphological rules is very complicated and understanding of such writing is difficult.

The system consists of two parts: syntactic analyzer and morphological analyzer. Purpose of the syntactic analyzer is to parse an input sentence, to build a parsing tree, which describes relations between the individual words within the sentence, and to collect information about the input sentence, which the system figured out during the analysis process. It is necessary to provide a grammar file to the syntactic and morphological analyzers. There must be recorded syntactic or morphological rules of particular natural language grammar. Syntactic analyzer also needs information about the grammar categories of the word-forms of natural language. It uses the information during analysis process.

Basic methods and algorithms, which we used to develop the system, are operations defined on features' structures; trace back algorithm (for morphological analyzer); general syntactic parsing algorithm for context free grammar and features' constraints method. Features' structures are widely used on all levels of analysis. We use them to hold various information about dictionary entries and information obtained during analysis. Each symbol defined in a morphological or syntactic rule has an associated features' structure, which we initially fill from the dictionary, or the system fill them by the previous levels of analysis. Features' structures and operations defined on them we use to build up features' constraints. With general parsing algorithm, it is possible to get a syntactic analysis of any sentence defined by a context free grammar and simultaneously check features' constraints, which may be associated with grammatical rules. Features' constraints are logical expressions composed by the operations, which we defined on the features' structures. We attach features' constraints to rules, which we defined within a grammar file. If the constraint is not satisfied during the analysis, then the system will reject current rule and the search process will go on. We can attach features' constraints also to morphological rules. However, unlike the syntactic rules, we can attach constraints at any place within a morphological rule, only not at the end. This speeds up morphological analysis, because the system checks constraints early and it rejects incorrect word-form's division into morphemes in a timely manner.

Formalism, which we developed for the syntactic and morphological analysis is highly comfortable for human. It has many constructions that make it easier to write grammar file. Morphological analyzer has a built-in preprocessor. It utilizes STL standard library. Program operates in UNIX and Windows operating systems. We can compile it and use in any other platform, which contains modern C++ compiler.

FEATURE STRUCTURES

A feature structure is a specific data structure. It essentially is a list of "Attribute - Value" type pairs. The value of an attribute (field) may be either atomic, or may be a feature structure itself. This is a recursive definition; therefore we can build a complex feature structure, with any level of depth of nested sub-structures. Feature structures are widely used in Natural Language Processing. They are commonly used:

1. To hold initial properties of lexical entries in the dictionary;
2. To put constraints on parser rules. Certain operations defined on feature structures are used for this purpose.
3. To pass data across different levels of analysis

We use following notation to represent feature structures in our formalism. List of "Attribute – Value" pairs is enclosed in square braces. Attributes and values are separated by colon ":". In example:

S = [A: V1
B: [C: V2]]

It is possible to use short-hand notation for constructing feature structures. We can rewrite above example this way:

T1 = [A: V1]
T2 = [C: V2]
S = [(S, T1) B: T2]

Content of the feature structures listed in the parenthesis at the beginning is copied to the newly constructed feature structure. Below is a fragment of a formal grammar for defining feature structures in our formalism:

```
<feature_structure> ::= "["[<initialization_part>] [<list_of_pairs>] "]"
<initialization_part> ::= "(" {<initializer>} ")"
<initializer> ::= <variable_reference> | <constant_reference>
<list_of_pairs> ::= { <pair> }
<pair> ::= <name> ":" <value>
<name> ::= <identifier>
<value> ::= "+" | "-" | <number> | <identifier> | <string> | <feature_structure>
. . .
```

There are several operations defined on feature structures to perform comparison and/or data manipulation. Mostly well-known operation defined on feature structures is unification. In addition to the unification, we have introduced other useful operations that simplify working on grammar files in practice. The result of each operation is a Boolean constant "true" or "false". Below is a list of all implemented operations and their semantics:

A := B (Assignment) Content of the RHS (Right Hand Side) operand (B) is assigned to the LHS (Left Hand Side) operand (A). Consequently, their content becomes equal after the assignment. The assignment operation always returns "true" value.

·A = B (Check on equality) This operation does not modify content of the operands. Result of the operation is "true" when both operands (A and B) have the same fields (attributes) with identical values. If there is a field in one feature structure, which is not represented in the second feature structure, or the same fields does not have an equal values, then the result is "false".

·A <== B (Unification) Unification returns "true", when the values of the similar field in each feature structure does not conflict with each other. That means, either the values are equal, or one of the value is undefined. Otherwise the result of the unification operator is "false". Fields, that are not defined in LHS feature structure and are defined in RHS feature structure are copied and added to the LHS operand. If there is an undefined value in LHS feature structure, and the same field in the RHS feature structure is defined, that value is assigned to the corresponding LHS feature structure field.

·A == B (Check on unification) Returns the same truth value as unification operator, but the content of operands is not modified.

Check on equality or unification operations ("=" and "==") may take multiple arguments. For instance:

$$X == (A, B, C)$$

Where X, A, B, and C are feature structures. Left hand side of an operation is checked against each right hand side argument that way. And the result is "true" only when all individual operations return "true", otherwise "false".

There is also a functional way to write operations. In example, we can write "equal(A, B)" instead of "A = B". Following functions are defined "equal" (check on equality), "assign" (assignment), "unify" (unification), "unicheck" (check on unification), "meq" (multiple equality checking), "muc" (multiple unification checking).

CONSTRAINTS

In our system, we use features' structures and operations defined on them to put constraints on parser rules. That makes parser rules more suitable for natural language analysis than pure CFG rules. We have generalized notation of constraint ([2]). Constraint is any logical expression built up with operations defined on features' structures and basic logical operations and constants: & (and), | (or), ~ (not), 0 (false), 1 (true).

Parser rules we can write following way: $S \rightarrow A_1 \{ C_1 \} A_2 \{ C_2 \} \dots A_N \{C_N\}$, where S is an LHS non-terminal symbol, A_i (I =1, … , N) are terminal or non-terminal symbols (for morphological analyzer only terminal symbols are allowed), and C_i (I = 1, … , N) are constraints. Each constraint is check as soon as all of the RHS symbols located before we match the constraint to the input. If a constraint evaluates to "true" value then parser will continue matching, otherwise if constraint evaluates to "false" parser will reject this alternative and will try another alternative. There is a features' structure associated with each (S and A_i) symbol in a rule. If a symbol is a terminal symbol, then we take initial content of its associated features' structure from the dictionary or from the morphological analyzer (for syntactic analyzer). We take content for a non-terminal symbols from the previous levels of analysis. We use constraints not only to check the correctness of parsing and not only to reduce unnecessary variants. We also use them to transfer data to a LHS symbol, thus move all necessary information to the next level of analysis. We can use assignment or unification operations for this purpose. To access a features' structure for particular symbol, we can use a path notation. We write a path using angle brackets. For example, <A> represents a features' structure associated with the A symbol. We can access individual fields by listing all path components in angle brackets.

We define the formal syntax for a constraint this way (fragment):

<constraint> ::= <constraint_term> "|"<constraint_term>
<constraint_term> ::= <constraint_fact> "&" <constraint_fact>
<constraint_fact> ::= ["~"] (<logical_constant> | "+" | "-" | <constraint_operation> | "(" <constraint_fact> ")")
<logical_constant> ::= "0" | "1"
<constraint_operation> ::= < constraint_operator> | <constraint_function>
<constraint_operator> ::= <constraint_argument> (":=" | "==" | "<==", "=") (<constraint_argument> | <list_of_constraint_arguments>)
<constraint_function> ::= <identifier> <constraint_function_arguments>
. . .

MORPHOLOGICAL ANALYZER

Purpose of morphological analyzer is to split an input word into the morphemes and figure out grammar categories of the word. We may invoke morphological analyzer manually or automatically by the syntactic analyzer.

We used special formalism to describe morphology of natural language and pass it to the morphological analyzer. There are two main constructions in the grammar file of morphological analyzer: morphemes' class definition, and morphological rules [10-12]. Morphemes' class definition is used to list all possible morphemes for a given morphemes' class. For example:

@M1 = {"morpheme_1" [… features …]"morpheme_2" [… features …]
. . . "morpheme_N" [… features …]}

It is possible to declare empty morpheme, which means that we may omit the morphemes' class in morphological rules. Below is formal syntax for morphemes' class definition:

<morphem_definition> ::= "@" <identifier> "= <list_of_morphemes> "}"
<list_of_morphemes> ::= <morpheme> { "," <morpheme> }
<morpheme> ::= <string> <feature_structure>

We define morphological rules following way:

word -> M_1 { C_1 } M_2 { C_2 } . . . M_N { C_N }

Where M_i are morpheme classes, and C_i (I = 1, … , N) are constraints (optional).

EXAMPLE OF GRAMMAR'S FILE COMPOSITION

Suppose, we wish to develop morphological analysis' program for nouns by morphological analyzer (ma); Firstly, we should fix morphemes' classes for nouns and enumerate them by their meetings in a noon. For the reason of example's simplifying, we will consider stems, number's signs and declension's signs only. Stems' classis consists of all nouns' stems. Class of number's signs consists of "eb", "n", "t" and ""(wide) morphemes. Declension's signs classis consist of declension's morphemes of all nouns. We should they pass to ma as starting information. For uniquely recognition of declension's category of a noun-form, we need to classify nouns' stems by attachment of declension's signs. For instance, non-compressed noun's stem ended by consonant. They attached declension's signs uniquely determine declensions of noun-forms. We must attach to such stem the feature (stem-type , "1"), where stem-type is the attribute and "1" is its value and it signifies non-compressed stem ended by a consonant. Then establishment of declension for such noun-forms is easy. We must compose the rule, which we can express so separate from a noun-form a stem, number's sign and declension's sign and if the noun-form coincide with founded morphemes completely and if stem-type = "1" and declension's sign = "i" then the noun-form has as declension's sign morpheme i and declension is nominative or declension's sign = "ma" and so on. The program will be:

noun-form-> stem {<noun-form stem> := <stem lex>}number {<noun-form number-lex> := <number lex>} declension {<noun-form declension-lex> := <declension lex> & (<stem-type> = "1" & <declension declension-sign> = "i" & <noun-form declension-sign> := "nominative")}.

The program can establish nominative declension for noun-forms, which have non-compressed stems ended with a consonant. To compose complete program, we must add to the program all cases such as other possible declension for the type of stems, other types of stems and rules for establishment of the number of noun-forms. noun-form designates non-terminal symbol for noun-forms. stem, number and declension are names of morpheme's classes, constraints are placed in figural scopes. We can assign to one feature another feature's value or textual constant. If the rule is satisfied then noun-form's feature gives concrete noun-forms' partitioning in morphemes and its morphological categories. Denotations in a rule are not restricted, but it is suitable to use meaningful denotation. It is obvious from the example, that composition of such program does not need the knowledge of programming. We must record such program in grammar's file.

SYNTACTIC ANALYZER

Purpose of syntactic analyzer is to analyze sentences of natural language and produce parsing tree and information about the sentence. In order to accomplish this task, syntactic analyzer needs a grammar's file and a dictionary (or it may use morphological analyzer instead of complete dictionary). We write grammar rules for syntactic analyzer like CFG rules. However, they may have constraints and symbol position regulators. We can write the rule according to these conventions:

S -> A_1 { C_1 } A_2 { C_2 } . . . A_N { C_N } ;

S -> A_1 A_2 . . . A_N : R { C } ;

Where S is an LHS non-terminal symbol, A_i (I = 1, … , N) are RHS terminal or non-terminal symbols, C and C_i (I =1, … ,N) are constraints, and R is a set of symbol position regulators. Position regulators declare order of RHS symbols in the rule, consequently making non-fixed word ordering. There are two types of position regulators:

1. $A_i < A_j$ means that symbol A_i must be placed somewhere before the symbol A_j .
2. $A_i – A_j$ means that symbol A_i must be placed exactly before the symbol A_j .

CONCLUSION

Described software tools we used for morphological and syntactic analyses of Georgian texts. All problems mentioned above were resolved. We simplified composition of grammar file by using macros with parameters. The software may be used for other natural languages. Method of composition of program file and initial data are the same.

The work was supported by GNSF(Georgian National Scientific Foundation), grant 09-184-1-120.

REFERENCES

1. Antidze J., Gulua N., Mishelashvili D., Nukradze L. On Complete Computer Morphological and Syntactic Analysis of Georgian Texts, Report of International Symposium – Natural Language Processing, Georgian Language and Computer Technologies, Institute of Linguistics of Georgian Academy of Sciences, Tbilisi, 2009
2. Antidze J. Formal Languages and Grammars, Natural Languages Computer Modeling, "Nekeri" , Tbilisi, 2009
3. Datukishvili K., Loladze M., Zakalashvili N. Georgian Language Processing (morphological level), Report of Symposium – Natural Language Processing, Georgian Language and Computer Technologies, Institute of Linguistics of Georgian Academy of Sciences, Tbilisi, 2003
4. Margvelani L. Machine Analysis System of Georgian Word Forms, Report of Symposium – Natural Language Processing, Georgian Language and Computer Technologies, Institute of Linguistics of Georgian Academy of Sciences. Tbilisi, 2003
5. Antidze J., Mishelashvili D. Instrumental Tool for Morphological Analysis of Some Natural Languages, Reports of Enlarged Session of the Seminar of IAM TSU, vol.19, Tbilisi, 2004
6. Antidze J., Melikishvili D., Mishelashvili D. Georgian Language Computer Morphology, Conference – Natural Language Processing, Georgian Language and Computer Technology, Tbilisi, 2004
7. Antidze J., Gulua N. On selection of Georgian texts computer analysis formalism, Bulletin of The Georgian Academy of Sciences, 162, N2, 2000
8. McConnell S. PC-PATR: Reference Manual, a unification based syntactic parser, version 1.2.2 http://www.sil.org/pcpatr/manual/pcpatr.html
9. Antworth E., McConnell S. PC-Kimmo Reference Manual, A two-level processor for morphological analysis, version 2.1.0 http://www.sil.org/pckimmo/v2/pc-kimmo_v2.html
10. Melikishvili D. System of Georgian verbs conjugation, Logos Press,Tbilisi, 2001
11. Melikishvili D. The Georgian Verb: A Morphosyntactic Analysis, Dunwoody Press, 2008
12. Melikishvili D. On Georgian Verb-forms Classification and Qualification Principles, Problems of Linguistics, 1, 2008

Надійшла до редакції 21.11.2010р.

ANTIDZE J. – PhD, Tbilisi state university, Tbilisi, Georgia.

GULUA N. – PhD, Sukhumi state university, Tbilisi, Georgia.