Proceedings of I. Vekua Institute
of Applied Mathematics
Vol. 56-57, 2006-2007

MACHINE TRANSLATION FROM GEORGIAN LANGUAGE INTO ANOTHER

Antidze J., Gulua N.

Iv. Javakhishvili Tbilisi State University
2 University Str., 0186 Tbilisi, Georgia
e.mail: jeantidze@yahoo.com

Abstract. In the article there is considered the problem of machine translation of text from Georgian to another language, an approach is proposed, using special method for morphological, syntactic and semantic analysis. Programming means are created which simplify whole process of translation.

Keywords and phrases: Machine translation, parsing, formal grammar, finite automaton, automaton with memory, feature structure, transfer.

AMS subject classification (2000): 68T50.

1. Introduction

Machine translation from one language into another is important problem, full resolution of which will resolve successfully many other problems of artificial intelligence as well. Automatic recognition of a text's content is basic difficulty on the road of the problem's resolution. For now, there is not such algorithm, which will fully resolve the problem. Because of the problem is simplified by splitting of text into sentences and considering each sentence separately. But it rests word's ambiguity yet and entire sentence may be ambiguous as well.

Word's ambiguity is widely propagated phenomenon in natural languages. Especially, it is difficult belles-lettres' translation. Therefore it is considered scientific-technical texts were word's ambiguity is comparatively less. For recognition of an ambiguous word into a sentence it is used context of the word. In most case context gives positive result. If the context does not get result then it is taken more frequently used meaning or all meanings.

Machine translation from one language into another contains the following stages:

1. Morphological analysis of words;

2. Syntactic analysis of sentence;

3. Construction of a sentence structure on target language corresponding to syntactic structure found in original sentence (transfer);

4. Composition of word forms of sentence in target language.

In most case the stages do not separated strictly one from another. It is possible parallel consideration of 1 and 2 stages. Resolution of ambiguity is considered in all parts. 1 and 2 stages call analysis of original language and 3 and 4 stages call synthesis. Let's consider each part separately for machine translation from Georgian language into another.

2. Morphological Analysis

It means recognition of each word of a sentence and establishing of morphological categories for them. Under full morphological analysis, we understand all possible splitting of a word-form in morphemes and establishment of morphological categories for it. It is widely spread the following ambiguity:

1. Graphical coincidence for different(by meaning) verb-forms in presence circle, which have the same root. For instance, verb-form "agebs", which may be signify loss many("agebs fuls"), build of plans("agebs gegmebs"), he corrects the bed("agebs logins") and so on;

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

3. In time of splitting verb-form, graphical coincidence different morphemes. For instance, "a" as 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-forms, we can not say, which morpheme we have, before have seen following two letters. This means, that Georgian verbs splitting in morphemes needs at least parsing algorithm for $LL(2)$ grammar [1-2] i.e. complete morphological analysis of Georgian words by finite automaton impossible.

In the case of second example, morphological analysis for verb-form "amoxsna" must give two different parsings: one for verb's infinitive and second for verb-form.

For this, we need nondeterministic algorithm. Deterministic algorithm can not give two different parses for the same word-form. And so deterministic algorithm not valid for complete morphological analysis of Georgian words. All morphological analysis for Georgian words are fulfilled by finite automaton or by deterministic algorithm.

From this follows, we must apply nondeterministic algorithm, for instance, from left to right in depth search algorithm with backtracking. As far, as backtracking take down the speed of the algorithm, we must find a method to reduce backtracking. It exists such possibility. We can exclude morphemes, which conflicts with found morphemes in a moment In other case, we can divide morphemes in classes so, that one representative of each classis may meet as maximum one times in a word-form.

Among morphemes of a verb-form, it is important roots. We can divide roots by classes so, that to fix morphemes, which can meet with one representative of each classis as maximum. All this reduce considerably numbers of backtracking. After splitting a word by morphemes, establishment morphological categories of the word is easy. We have developed instrumental tool by which realized morphological analysis of Georgian words [3-5].

3. Software Tools

The "Software Tools for Morphological and Syntactic Analysis of natural Language Texts" is a software system designed for natural language texts processing. The system is used to analyze syntactic and morphological structure of the natural language texts. Specific formalisms, that have been worked out for this purpose, allow us to write down syntactic and morphological rules defined by particular natural language grammar [5].

These formalisms represent a new, complex approach, that solves some of the problems connected with the natural language processing. A software system has been implemented according to these formalisms. Syntactic analysis of sentences and morphological analysis of word-forms can be done within this software system. Several special algorithms were designed for this system. To use formalisms, which are described in [6-7] is very difficult for Georgian language.

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 all important information about the input sentence, which has been figured out during the analysis process.

It is necessary to provide a grammar file to the syntactic analyzer. There must be written syntactic rules of particular natural language grammar in that file. Syntactic analyzer also needs information about the grammar categories of the word-forms of natural language. Information about the grammar categories of the word-forms is used during the analysis process.

However, it may be quite difficult to include all of the word-forms from the natural language into a dictionary file. To avoid this problem and to reduce size of dictionary file, morphological analyzer is used. Morphological analyzer uses a dictionary file of unchanged parts of words. Therefore this file will be considerably smaller, because many word-forms can be produced by single unchanged part of word.

The morphological analyzer also needs its own grammar file. According to the specific formalism, morphological rules of natural language must be written in that grammar file. An input word is divided into morphemes when applying these rules and important information about the grammar categories of word-form can be deduced during the analysis.

An input sentence is passed to syntactic analyzer. Syntactic analyzer passes each word from the sentence to morphological analyzer. Morphological analyzer will analyze the words according to the rules from the grammar

file, using a dictionary of words' unchanged parts. After the successful analysis each word-form will obtain information about its grammar categories, and this information will be returned to the syntactic analyzer. At the end syntactic analyzer will try to parse the sentence according to the rules from the syntax file.

Basic methods and algorithms, which were used to develop the system are operations defined on the feature structures, trace back algorithm (for morphological analyzer), general syntactic parsing algorithm and feature constraints method.

Feature structures are widely used on all level of analysis. As an abstract data types they are used to hold various information about dictionary entries. Each symbol defined in a morphological or syntactic rule has an associated feature structure, which is initially filled from the dictionary, or it is filled by the previous levels of analysis.

Feature structures and operations defined on them are used to build up feature 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 feature constraints, that may be associated with grammatical rules. Feature constraints are logical expressions composed by the operations, which are defined on the feature structures.

Feature constraints can be attached to rules, which are defined within a grammar file. If the constraint is not satisfied during the analysis, then the current rule will be rejected and the search process will go on. Feature constraints also can be attached to morphological rules. However, unlike the syntactic rules, constraints can be attached at any place within a morphological rule, not at the end only.

This speeds up morphological analysis, because constraints are checked as soon as they are met in the rule, and incorrect word-form divisions into morphemes will be rejected in a timely manner.

Formalisms that were developed for the syntactic and morphological analyzers are highly comfortable for human. They have many constructions that make it easier to write grammar files. Morphological analyzer has a built-in preprocessor, which has a capability to process macros.

The software system is written in c++ programming language standard. It utilizes STL standard library. The system operates in UNIX and Windows operating systems. Although, it could be compiled and used in any other platform, which contains modern c++ compiler.

4. 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 ":". For 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:

$\langle feature-structure\rangle$ ::= "[" [$\langle initialization-part\rangle$] [$\langle list-of-pairs\rangle$] "]"

$\langle initialization-part\rangle$ ::= "(" {$\langle initializer\rangle$} ")"

$\langle initializer\rangle$ ::= $\langle variable-reference\rangle$ | $\langle constant-reference\rangle$

$\langle list-of-pairs\rangle$ ::= {$\langle pair\rangle$}

$\langle pair\rangle$ ::= $\langle name\rangle$: $\langle value\rangle$

$\langle name\rangle$ ::= $\langle identifier\rangle$

$\langle value\rangle$ ::= " + " | " − " | $\langle number\rangle$ | $\langle identifier\rangle$ | $langlestring\rangle$ | $\langle feature-structure\rangle$

...

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 composing of 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".

- $A\langle == B$(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 example:

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

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. For 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) and "muc"(multiple unification checking).

5. Constraints

In our system feature structures and operations defined on them are used 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 [8].

Constraint is any logical expression built up with operations defined on feature structures and basic logical operations and constants: &(and), | (or),˜(not), $0(false), 1(true)$. Parser rules are written following way

$$S \to A_1\{C_1\}A_2\{C_2\}\ldots A_N\{C_N\}$$

Where S is an LHS nonterminal symbol, $A_i(I = 1, \ldots, N)$ are terminal or nonterminal symbols (for morphological analyzer only terminal symbols are used) and $C_i(I = 1, \ldots, N)$ are constraints.

Each constraint is check as soon as all of the RHS symbols located before the constraint are matched 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 it will try another alternative.

There is a feature structure associated with each(S and A_i) symbol in a rule. If a symbol is a terminal symbol then initial content of its associated feature structure is taken from the dictionary or from the morphological analyzer(for syntactic analyzer).

Content for a nonterminal symbols is taken from the previous levels of analysis. Constraints are used not only to check the correctness of parsing and reduce unnecessary variants. They are also used to transfer data to a LHS symbol, thus move all necessary information to the next level of analysis.

Assignment or unification operations can be used for this purpose. To access a feature structure for particular symbol, a path notation can be used. Path is written using angle brackets.For example, $\langle A \rangle$ represents a feature structure associated with the A symbol. Individual fields can be accessed by listing all path components in angle brackets.

The formal syntax for a constraint is defined this way(fragment):

$\langle constraint \rangle$ ::= $\langle constraint{-}term \rangle$" | "$\langle constraint{-}term \rangle \langle constraint{-}term \rangle$ ::= $\langle constraint - fact \rangle$"&"$\langle constraint - fact \rangle$

$\langle constraint - fact \rangle$ ::= ["~ "] ($\langle logical - constant \rangle$ | " + " | " − " | $\langle constraint - operation \rangle$ | "(" $\langle constraint - fact \rangle$ ")")

$\langle logical - constant \rangle$ ::= "0" | "1"

$\langle constraint - operation \rangle$ ::= $\langle constraint - operator \rangle$ | $\langle constraint - function \rangle$

$\langle constraint - operator \rangle$ ::= $\langle constraint - argument \rangle$(" := " | " == " | " <== ", " = ")

($\langle constraint - argument \rangle$ | $langlelist - of - constraint - arguments \rangle$)

$\langle constraint - function \rangle$ ::= $\langle identifier \rangle$ $\langle constraint - function - arguments \rangle$

...

6. Morphological Analyzer

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

Special formalism has been created 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: morpheme class definition, and morphological rules. Morpheme class definition is used to list all possible morphemes for a given morpheme class. For example:

"@"$M1$ M1=
{
"morpheme_1" [. . . " features " . . .]
"morpheme-2" [. . . " features " . . .]
. . .
"morpheme-N" [. . . " features " . . .]
}

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

$\langle morphem - definition\rangle$ ::="@"$\langle identifier\rangle$
" = ""{"$\langle list - of - morphemes\rangle$"}"
$\langle list - of - morphemes\rangle$ ::= $\langle morpheme\rangle${","$\langle morpheme\rangle$}
$\langle morpheme\rangle$ ::= $\langle string\rangle\langle feature - structure\rangle$

Morphological rules are defined following way:

word $\rightarrow M_1\{C_1\}M_2\{C_2\}\ldots M_N\{C_N\}$

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

7. 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 file and a dictionary(or it may use morphological analyzer instead of complete dictionary). Grammar rules for syntactic analyzer are written like CFG rules. But they may have constraints and symbol position regulators. The rule can be written according to these constructions:

$S \rightarrow A_1\{C_1\}A_2\{C_2\}\ldots\ A_N\{C_N\}$
$S \rightarrow A_1A_2\ldots\ A_N\ R\{C\}$

Where S is an LHS non-terminal symbol, $A_i(I = 1\ \ldots\ N)$ are RHS terminal or non-terminal symbols, C and $C_i(I = 1\ \ldots\ N)$ 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.

8. Example of Special Program Composition

Suppose we will develop morphological analysis of nouns by instrumental tool(IT). Firstly, we should fix morphemes' classes for nouns and enumerate them by it meeting in a noon. By reason of example's simplifying, we

will consider stems, number signs and declension signs only. Stems class consist of all noun's stems. Number signs' class consist of "eb", "n", "t" and ""(wide) morphemes. Declension's signs class consist of all nouns' declension morphemes. We should they pass to IT as starting information. For uniquely recognition declension category of a noun-form, we need to classify noun's stems by attachment of declension signs, for instance, non-compressed nouns' stems, which end by consonant. They attached declension signs uniquely determine declensions of noun-forms.

We must attach to such stem the feature(stem-type equal to "1"), where stem-type is the attribute and "1" is its value and signify non-compressed stem ended by a consonant. Then establishment of declension for such noun-forms is easy. We must compose the rule, which is expressed so:

separate from a noun-form stem, number's sign and declension sign and if the noun-form coincide with founded morphemes completely and if stem-type equal to "1" and declension sign equal to "i" then the noun-form has as declension sign morpheme i and declension equal to nominative or declension sign equal to "ma" ... and so on. The special program will be:

$noun - form \rightarrow stem\{\langle noun - form\ stem\rangle := \langle stemlex\rangle\ \}$number $\{\langle noun - form\ number - lex\rangle := \langle number\ lex\rangle\ \}$ declension $\{\ \langle noun - form\ declension - lex\rangle := \langle declension\ lex\rangle\ \&\ (\langle stem - type\rangle =$ "1" $\&\ \langle declension\ declension - sign\rangle =$"i" $\&\ \langle noun - form\ declension - sign\rangle :=$ "nominative")$\}$.

The program can establish nominative declension for noun-forms which have non-compressed stems ended with a consonant. To compose complete special program, we must add to the program all cases such is 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 features gives concrete noun-forms' partitioning by 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. Such program is recorded in grammar file.

9. Semantic Analysis

Complications of semantic analysis are caused by peculiarities of sentences. Let discuss some of them:

1. A sentence can be true or false. The total of inner corners of a rectangle is equal to 360, whether it is true or false can be proved logically, but the truth of the sentence which describes some historical event

can not be proved logically. It must be accepted by historians as a fact or denied by them as such;

2. A sentence can be universal or typical, e.g. All humans are mortal is universal, but All men like alcohol is not universal, but is typical

3. A sentence can be trustworthy or unbelievable;

4. A sentence can be rare or approximately trust-worthy;

5. A sentence can be homonymous in sense of its meaning.

We can present the peculiarities by constraints on feature structure [8] modifying syntactic rules by adding new constraints and express them by our tools.

10. Transfer

After receiving the syntactic structure of a sentence (syntactic tree) in original language, we must compose corresponding syntactic structure in the target language. For this, we must have for each syntactic rule corresponding syntactic and semantic rule in target language. In this case, we can compose syntactic tree in target language and form each word using information in the syntactic tree and using rules of target language morphology.

11. Conclusion

In the article, we have outlined marginal problems of machine translation from one language into another, presented problems of automatic analysis of Georgian texts and our approach of their resolution.

REFERENCES

1. Aho A.V., Ulman J.D. The Theory of Parsing, Translation and Compiling, vol. 1, Prentice-Hall Series in Automatic Computation, *Prentice-Hall, Inc., Englewood Cliffs, N.J.*, 1972, 420 p.

2. Antidze J. Formal Languages and Grammars, Natural Language Computer Modeling, 2007, Nekeri, Tbilisi, http://fpv.science.tsu.ge/mon-2007.pdf

3. Antidze J., Mishelashvili D. Software tools for morphological and syntactic analysis of natural language texts,*Internet Academy, Georgian Electronic Scientific Journals, Computer Sciences and Telecommunications,* **1**, 12 (2007), 50-58, ISSN 1512-1232, http://gesj.
internet-academy.org.ge/gesj_articles/1345.pdf.

4. Antidze J., Mishelashvili D. Instrumental tool for morphological analysis of some natural languages, *Rep. Enlarged Sess. Semin. I. Vekua Inst. Appl. Math.*, **19** (2004), 15-19.

5. Melikishvili D. Conjugation system of the Georgian verb. *Logos Press, Tbilisi*, 2001, 362 p. (in Georgian).

6. McConnell S. PC-PATR Reference Manual, a unification based syntactic parser. version 1.2.2., 20 p. http://www.sil.org/pcpatr/manual/pcpatr.html.

7. Antworth E., McConnell S. PC-Kimmo Reference Manual, a two-level processor for morphological analysis. version 2.1.0., 57 p. http://www.sil.org/pckimmo

8. Antidze J., Gulua N. On selection of Georgian texts computer analysis formalism, Bull. Georgian Acad. Sci. **162**, 2 (2000), 54-57, http://fpv.science.tsu.ge/pub5.mht.

Received 18.06.2007; revised 15.10.2007; accepted 20.12.2007.