CFG Based Grammar Checker for Latvian

Daiga Deksne
Tilde
Vienības gatve 75a, Riga, Latvia
LV1004
daiga.deksne@tilde.lv

Raivis Skadiņš
Tilde
Vienības gatve 75a, Riga, Latvia
LV1004
raivis.skadins@tilde.lv

Abstract

This paper reports on the implementation of the Latvian grammar checker. It gives a brief introduction of the project scope – Latvian language, the previous implementation of the grammar checker and its limitations. Then, it describes the proposed approach. This paper also describes the Latvian parser used for this project and the quality measurement methods used for the quality assessment of the grammar checking system. Finally, the current state of the grammar checker work is presented.

1 Introduction

The grammar checker described in this paper is not the first implementation of a Latvian grammar checker. The first Latvian and Lithuanian grammar checkers were implemented in 2004 (Mackevičiūte, 2004). Grammar checkers where implemented using an advanced pattern matching. There were almost 200 rules such as:

- If there is any verb in the imperative mood followed by an adverb 'lūdzu' (please), then suggest inserting comma between these words.
- If there is a noun in the nominative followed by a (i) comma, (ii) preposition "uz" and (iii) pronoun "kurš" in the singular genitive or plural dative AND genders of noun and pronoun are different; then suggest changing the gender of the pronoun to be equal with the gender of the noun.

These rules highlighted many grammar errors, but the grammar checker had many deficiencies; the most significant were:

- This format did not describe long distance errors and errors that describe complex syntactic structures. Only patterns matching near words were allowed.
- Many rules had to be disabled because they matched false errors caused by high morphological ambiguity.
- The pattern matching algorithm was quite slow and each new grammar rule made the grammar checker slower and slower.

All the obstacles mentioned above led to the work presented in this paper. A new Latvian grammar checker has been built based on more powerful techniques.

2 Chosen approach

2.1 Main principles

As Latvian is highly inflected language with a high morphological ambiguity there are many long distance agreements between words and phrases in a sentence for which we need a deep syntactic analysis of phrases and sentence to find possible errors. The new implementation of the Latvian grammar checker is based on a parser. The parser works with two sets of rules:

- Rules describing Latvian grammar, e.g. correct syntactic structures (G rules);
- Rules describing grammar errors (E rules).

If parser would work only with G rules it would fully parse grammatically correct sentences and partly parse ungrammatical sentences and also sentences whose syntactic structure is too complex. For example, if we parse the Latvian text "Manam piemēram ir jābūt skaidram. Piemēram es saprotu to." (My example must be clear. For example I understand it) we get a parse as in Figure 1. The first sentence is fully parsed therefore we can consider it to be grammatical, the second sentence is only partially parsed therefore it is either ungrammatical or it is too complex to be fully parsed with a current set of G rules.

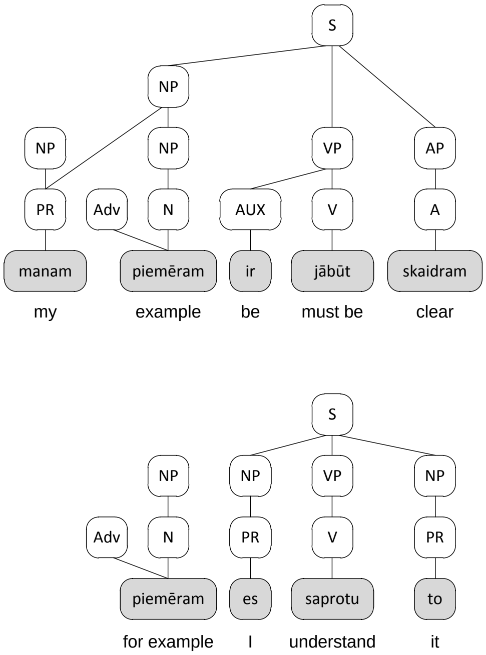

Figure 1. Result of parsing when parsing with G rules only.

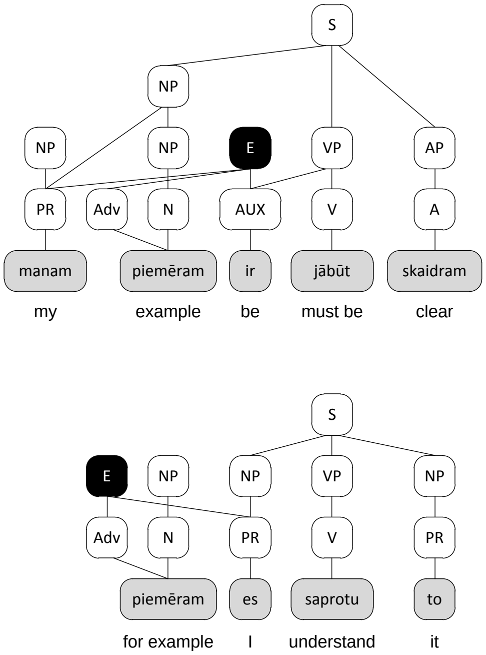

Figure 2. Result of parsing when parsing with both G and E rules.

If we add rules that also describe syntactic errors (E rules) we get a parse as in Figure 2. We get a similar result as before. The second sentence still is not fully parsed, but the parser has applied an error rule which finds the adverb ‘piemēram’ followed by pronoun. The parser has applied a similar error rule in the first sentence too. We can ignore this error rule in the first sentence because we know that that sentence is fully parsed (grammatical). But an error rule in the second sentence really marks a grammar error as the sentence (or phrases containing words marked by error rule) has not been fully parsed.

2.2 Parser

There are some requirements for the parser in order to use it to find grammar errors in the way described above. (i) The parser must be robust and return partial parses if the sentence cannot be fully parsed; (ii) The parser must be able to return all possible parses not only the one. As seen in Figure 2 error rules are not a part of parse trees; (iii) The parser must mark as correct only syntactic structures which really are correct; (iv) As we are working with Latvian, the parser rules must be powerful enough to deal with high morphological variance and ambiguity, word agreement and a rather free word order.

For the purposes of grammar checking we used the Latvian parser developed for machine translation purposes (Skadiņš *et al.*, 2007). The parser is using adapted CFG grammar (Chomsky, 1956) and it is based on the CYK algorithm (Younger, 1967) which allows partial parsing if the sentence cannot be fully parsed. The CYK algorithm is extended to support attributes for both terminals and non-terminals.

2.3 Rule format

As Latvian is a morphologically rich language Latvian grammar cannot be described with simple CFG rules like NP →N; NP →N N; S→NP V NP. The CFG used in the Latvian parser uses attributes for terminal and non-terminal symbols. For example, the noun phrase NP has attributes number, gender, case, person and some more. The error rules operate with terminals and phrases which were created with correct grammar rules. In the rule body there are usually some agreement or disagreement statements between attributes of several in itself correct phrases. There also might be an attribute comparison with an exact value. Also, lexical parts might figure in such rules. Often there is a correct grammar rule with the same right side constituents as in some error rule, only the comparison operators are different. See sample of a correct grammar and an error rule in Figure 3. The error rules have a section where the correct attribute values are assigned and instructions for suggestion generation are given.

```
NP -> attr:CAP main:NP
      Agree(attr:CAP, main:NP, Case,
Number, Gender)


ERROR-1 -> attr:CAP main:NP
      Disagree(attr:CAP,main:NP,
Case, Number, Gender)
GRAMMCHECK MarkAll
      attr:CAP.Gender=main:NP.Gender
      attr:CAP.Number=main:NP.Number
      SUGGEST(attr:CAP+main:NP)
```

Figure 3. Error and correct grammar rules.

If all comparison operators in the error rule are true, it does not guarantee that this error will be flagged as seen in Figure 2. For an error rule to succeed, the phrase it covers must be larger than the phrase for which the correct grammar rule works.

We also have a second grammar containing only error rules. It does not rely on correct grammar phrases. Capitalization and incorrect writing style errors enclose shorter phrases often with exact lexical values. The CapPattern operator defines the correct capital/noncapital letter usage in phrases with special meaning like organization, institution names, country names, job titles, etc. (See Figure 4). If the capitalization pattern is different for a phrase in the text, an error rule is triggered.

```
ERROR-14 -> attr:N attr:G main:N
      attr:N.Case==genitive
      attr:N.Number==singular
      attr:G.AdjEnd==definite
      main:N.Number==plural
      Agree(attr:G, main:N, Case,
Number, Gender)
      CapPattern fff
  LEX Amerika savienots valsts
```

Figure 4. Capitalization error rule.

3 The grammar checker architecture

The grammar checking system consists of separate components each having its own task. Most of them must be called in a certain order as each component relies on data structures prepared by the previous component.

The incoming text is split into separate token objects and sentence boundaries are detected in a tokenizer module. Subsequent components work only with a sentence, not with all incoming text at once. One of the following token types is assigned to every token object: word, abbreviation, punctuation and numeric. In a simple error location module simple formatting errors are located using regular expressions. The analyzer module adds morphological analysis to every token. The parser component performs parsing using a given rule set. The parse walker component extracts the error trees from the parse result matrix and generates suggestions for error fixing. Results from this component and from the simple error locator are passed to the result preparation module which merges results and returns to a calling application.

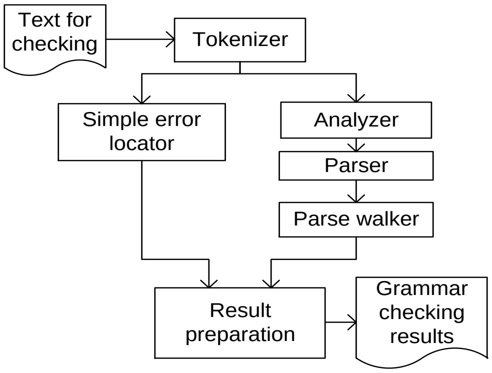

Figure 5. Grammar checker architecture.

4 The quality measuring methods

Test and development corpora are prepared to measure the quality of grammar and to have an assurance that the grammar checker works with approximately the same quality on any text. The test corpus is used only to measure the current quality of the grammar checker and rule developers do not see its content; the development corpus is also used in the process of tuning the rules.

Both corpora contain a variety of texts. About an equal amount of texts from every type are included in both corpora. We assume that potential users of the grammar checker will want to use it for checking grammar in the following types of texts: high school student essays, university student papers, blogs (qualitative, but not edited), e-mails (qualitative, but not edited), non-edited marketing texts, non-edited written texts from non-native Latvian speakers with good Latvian language knowledge, news texts, draft of some project tender (not edited), the works of new (amateur) writers, texts from the specialists in certain fields (teacher of physics, programmer, doctor, lawyer, geographer, psychologist, …)

The information about errors and expected corrections for each sentence is stored in a Golden Standard. The Golden Standard can be updated in two ways:

- A human annotator marked the sentences with error types prior to the grammar checking in the development corpus;
- After the grammar checking of both corpora, results are compared with the Golden Standard. Previously unseen cases are given to the human evaluator for the evaluation. The evaluator checks whether the error found by the grammar checker and the suggested correction is correct or not. Based on this information the Golden Standard is updated.

Several measurement values – recall, precision, f-measure, confidence interval for the precision – are calculated for every error type. The value of recall shows the possibility of finding all existing errors in the text. The recall is a number of correctly found errors (of type x) divided by number of errors (of type x) in corpus.

$$R(x) = tp(x)/(tp(x)+fn(x))$$

The value of precision shows the possibility of correctly finding errors in the text. The precision is a number of correctly found errors (of type x) divided by number of correctly and incorrectly found errors (of type x) in corpus.

$$P(x) = tp(x)/(tp(x)+fp(x))$$

Improvement of grammar rules is done based on the development corpus, the Golden Standard and evaluation results; the recompiled grammar is used for repeated evaluation and elaboration.

The test corpus contains 4814 sentences, the development corpus - 9364 sentences. Recall is given only for the development corpus, as the test corpus was not previously marked.

5 Results

So far our grammar checking system works with two grammars. The first one contains rules describing incorrect capitalization patterns in phrases and style errors. It contains 260 rules. The second is made of a set of 477 syntactically correct constructions describing rules and 237 error rules. Errors are classified with 21 error types. Precision and recall measures for eight most common error types are seen in Table 1.

The recall and precision values might be influenced by the fact that a sentence can contain several errors. Human evaluator is marking sentence with only a single error type. The grammar checking system is also selecting a single error per sentence – the one which covers the largest phrase. The error types of the human evaluator and the grammar checking system might not match.

Error type	**Recall**	**Precision**	
	Dev. corp	Dev. corp.	Test. corp.
Agreement between words	0.247	0.543	0.426
Punctuation error at the end of sentence	0.240	0.957	—
Words must be written together	0.761	0.962	1.000
Comma error in insertions	0.563	0.913	0.892
Comma error in participial phrase	0.427	0.704	0.660
Wrong writing style	0.397	1.0	0.950
Comma error in equal parts of sentence	0.140	0.773	0.583
Comma error in sub clause	0.329	0.773	0.758
All error types	**0.290**	**0.833**	**0.710**

Table 1. Grammar checker results for development and test corpus.

The developed grammar checker is integrated in Microsoft Word and OpenOffice Writer text editors, it works as a background process and it is fast enough for real everyday use. An evaluation of user satisfaction showed that users find it helpful. The evaluation also showed that users prefer a grammar checker with a high precision rather than a high recall.

Reference

Chomsky, N. 1956. Three models for the description of language. Information Theory, IEEE Transactions 2 (3): 113–124.

Mackevičiūtė, J. 2004. Lithuanian morphological analysis system and grammar checker: Tilde's technologies in practice. In Proc. HLT'2004, Riga, Latvia

Skadiņš R., Skadiņa I., Deksne D., Gornostay T. 2007. English/Russian-Latvian Machine Translation System. In Proc. HLT'2007, Kaunas, Lithuania

Younger, D. 1967. Recognition and parsing of context-free languages in time n3. Information and Control 10(2): 189–208.