Extended CFG Formalism for Grammar Checker and Parser Development

Daiga Deksne, Inguna Skadiņa, and Raivis Skadiņš

Tilde, Riga, Latvia
{daiga.deksne,inguna.skadina,raivis.skadins}@tilde.lv

Abstract. This paper reports on the implementation of grammar checkers and parsers for highly inflected and under-resourced languages. As classical context free grammar (CFG) formalism performs poorly on languages with a rich morphological feature system, we have extended the CFG formalism by adding syntactic roles, lexical constraints, and constraints on morpho-syntactic feature values. The formalism also allows to assign morpho-syntactic feature values to phrases and to specify optional constituents. The paper also describes how we are implementing the grammar checker by using two sets of rules – rules describing correct sentences and rules describing grammar errors. The same engine with a different rule set can be used for the different purposes – to parse the text or to find the grammar errors. The paper also describes the implementation of Latvian and Lithuanian parsers and grammar checkers and the quality measurement methods used for the quality assessment.

Keywords: parsing, grammar checking, inflected languages.

1 Introduction

Proofing tools have been in development for a rather long time. Tools for checking spelling are available for many languages in different word processing applications as well as other natural language applications. However, a more complicated task for computers is grammar checking. Due to the high ambiguity of languages, grammar checking tools are available for a rather small number of languages. Moreover, even grammar checking tools for the English language only allow for the correction of certain types of errors.

The problem becomes even more complicated when it concerns highly inflected languages that have a rather free word order. Only a few grammar checkers have been developed for such languages (e.g., there are several grammar checkers for the Russian language).

In this paper, we present a framework for grammar checking that is derived from a context-free grammar (CFG) formalism. A classical CFG performs poorly on inflected languages, e.g., large numbers of non-terminals are necessary for representation of morpho-syntactic features, as well as parser output usually consists of many parse trees. Thus different syntactic formalisms derived from CFG (e.g., Generalized Phrase

A. Gelbukh (Ed.): CICLing 2014, Part I, LNCS 8403, pp. 237–249, 2014.

Structure Grammar introduces mechanism for feature passing [10], Definite Clause Grammar expresses grammar as clauses of first-order predicate logic [19]) have been developed. In addition many syntactic formalisms that adopt phrase structure are proposed: for instance, Head-Driven Phrase Structure Grammar adopts the basic phrase structure syntax through unification of feature structures [20], Lexical Functional Grammar use constituent structure together with feature structure for syntax representation [14], Augmented transition network formalism [4] realizes unification through recursive transition network.

In this paper we propose to extend CFG by adding morpho-syntactic features and syntactic roles, and by introducing two operators - constraint checking operator and assignment operator. Additionally, rules for grammar checking are divided into two sets – one rule set for parsing and recognizing correct patterns and another rule set for error detection and correction. Our grammar checking system allows to correct 23 types of errors, including syntactic errors, style errors, and capitalization errors. For inflected languages, the most important groups of errors are word agreement errors and errors that are related to punctuation in specific constructions.

The developed framework is used to implement grammar checkers for two languages of the Baltic language group - Latvian and Lithuanian. The evaluation results for these languages are presented and discussed in this paper. In addition, we also demonstrate how the developed framework can be used for grammar checking of other inflected languages, e.g., the Slavic language group.

2 Related Work

The grammar checking problem has been actual since the 1970s, when language technologies obtained their intelligence. The first grammar checkers checked punctuation and style inconsistencies. In the early 80s, grammar checkers were released for personal computers, and, soon afterwards, grammar checkers that could detect writing errors beyond simple style errors were developed. Among grammar checkers we would like to mention the grammar checker in Word97 for English [11], the rule-based system for Dutch [27], ReGra for Brazilian Portuguese [16], first grammar checker for Latvian [7], grammar checkers for Swedish [1], [8], [22], German [23], and Arabic [24].

Different approaches have been used for grammar checking; the most popular being rule-based (e.g., constraint grammar, context-free grammar), statistical [2], [13], [25], and hybrid [8], [9], [28].

The rule-based approach usually uses manually created rules that can be easily modified, added, or removed. Such rules are linguistically motivated. However, it is not easy to maintain larger systems. One popular approach is Constraint Grammar (CG) formalism, which was originally designed by Fred Karlsson [15] for grammar-based parsing. However, CG parser can be used not only to tag a sentence with surface syntactic functions, but also to mark possible grammar errors. It has been used for the detection of syntactic errors in Swedish [1], [5] and Norwegian [12].

LanguageTool grammar checker[1] [17] is a rule-based open source grammar checking system that is a plug-in for *OpenOffice.org*. Currently, it supports 29 languages. However, support significantly varies from language to language.

Despite a long history of development of grammar checkers, there is a lot of space for improvements where it concerns language coverage and algorithms as it is demonstrated for English [18].

3 Extended Context-Free Grammar Formalism

Extended context-free grammar formalism is derived from CFG. Similar to CFG, our formalism contains a description of phrase structure. However, it usually also has a rule body consisting of constraints, lexical restrictions, and value assignment/ inheritance statements.

3.1 Structure of the Rules

Context-free grammar, formalized independently by Chomsky [6] and Backus [3], is defined as a 4-tuple (P, N, T, S) with the following components:

- **P** is a set of grammar rules or productions (i.e., items of the form X → a, where X is a non-terminal symbol, and *a* is a string of terminal and non-terminal symbols)
- **N** is the set of non-terminal symbols (i.e., grammatical or phrasal categories)
- **T** is the set of terminal symbols (i.e., words of the language)
- **S** is a designated start symbol, normally interpreted as representing a full sentence

Classical CFG is powerful and efficient for describing sentence structures of analytic languages that convey grammatical relationships without the use of inflectional morphemes. However, it is not so efficient in describing the sentence structure of synthetic languages that use inflectional morphemes and have a quite free word order. It is especially difficult to describe agreement (or disagreement) between words or phrases. Therefore, we have introduced morpho-syntactic properties to CFG, have allowed non-terminals to inherit these properties from their constituents, and have used these properties to restrict rules.

We use terminals and non-terminals in the same way as CFG does. However, on the right side of the production rule, syntactic roles are added to each constituent as shown in (1) for noun phrase NP:

NP -> attr:NP main:N (1)

In each rule, the syntactic role *main* is mandatory for the head constituent, and other possible roles include *subj, obj, mod,* etc.

The body of the rule usually contains some constraints and some assignments. The constraints are used to restrict the application area for the rule and to avoid overgeneration. They are realized through morpho-syntactic properties of terminals and

[1] `https://www.languagetool.org/`

non-terminals. The set of properties and property values is language specific. For Latvian and Lithuanian, we use 26 properties, such as, number, case, gender, etc. Table 1 provides a summary of comparison and assignment operators.

Table 1. Comparison and assignment operators

Operation	Sample	Explanation
Strict comparison with a constant	`attr:P.Case==dative`	The case of the pronoun (P) must be dative
Comparison with a constant when the property's value is defined	`attr:P.Case===dative`	If the case of the pronoun is defined, it must be dative
Comparison with a constant for inequality	`main:VP.Person!=III`	The person of the verb phrase (VP) must not be 3rd
Strict comparison of property values for two right side constituents	`mod:P.Case==main:NP.Case`	The case values of pronoun (P) and noun phrase (NP) must be equal
Comparison of property values for two right side constituents when their values are defined	`mod:P.Case===main:NP.Case`	The case values of pronoun (P) and noun phrase (NP) must be equal, if the case values are defined
Comparison for inequality of two right side constituents	`mod:P.Case!=main:NP.Case`	The case values of pronoun (P) and noun phrase (NP) must be different
Assignment of constant	`NP.Person=III`	The 3rd person is assigned to the left side noun phrase
Inheritance/ assignment of property values of right side constituents	`NP.Case=main:NP.Case`	The case of the right side noun phrase is assigned to the left-side noun phrase

Two functions are introduced that allow to check agreement between constituents with a single statement. Function *Agree(item1, item2, property-1, property-2, …, property-n)* allows to check whether values of *property-1, …, property-n* are equal for constituents *item1, item2*. For instance, (2) checks the agreement of noun (N) and adjective (A) in case, number, and gender.

Agree(attr:A, main:N, Case, Number, Gender) (2)

Similarly, the *Disagree(item1, item2, property-1, property-2, …, property-n)* function checks whether at least one of the properties *property-1, property-2, …, property-n* differs between *item1* and *item2*. This is especially useful for grammar checking. For instance, in the case of error in a noun phrase, there could be a disagreement between noun and adjective in gender, case, or number (3).

Disagree(attr:A, main:N, Gender, Number, Case) (3)

With a LEX statement the terminal symbol lexical values – base forms – can be specified. The number of words in a LEX statement must be equal to the number of right side constituents in the rule. The '*' symbol is used to allow any value for the constituent. There can be several LEX statements in a rule, as shown in Fig. 1, where the adverbial phrase (ADVP) consists of two adverbs (R), and the first adverb could only be either *pavisam* ('entirely') or *īpaši* ('especially').

```
ADVP -> ad:R main:R
 LEX pavisam *
 LEX  paši *
```

Fig. 1. Sample of rule with lexical constraints

3.2 Rule Specifics for Grammar Checking

For grammar checking, we also introduce error rules. In error rules, the left side non-terminal is in the form 'ERROR-id'. All rules that describe the same type of error have the same id. Error rules do not contain value assignment operators which assign values to the left side non-terminal; they contain only constraint expressions (Fig. 2).

```
ERROR-1 -> attr:A main:N
 Disagree(attr:A,main:N, Case, Number, Gender)
```

Fig. 2. Sample of error rule describing disagreement between noun N and adjective A in case, number, or gender

The error description is followed by correction suggestion part of the rule. It starts with the label "GRAMCHECK" and is followed by the markup operator that tags an error in the phrase. Operator *MarkAll* tags the whole phrase, while operator *Mark(some right side constituent[+other right side constituent]*)* tags the part of the phrase represented by the right side constituent(s). Property assignment statements allow for the changing of the properties of the right side items and are used for generating suggestions.

Finally, the statement SUGGEST is used to form correct output (concatenated with '+'). An example of the grammar checking rule is shown in Fig. 3.

The error rules may contain phrases that are created with parsing rules (e.g., noun phrase, adjective phrase, etc.), and there usually are some agreement or disagreement statements (between properties of several phrases that are correct within themselves) in the body of the error rule.

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

Fig. 3. Grammar checking rule that corrects disagreement between noun phrase (NP) and adjective phrase (AP) in case, number, or gender

Error rules are often coupled with rules describing correct grammar, i.e., there is a correct grammar rule with the same right side constituents as some error rule, and only the constraint operators differ (see Fig. 3 and Fig. 4).

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

Fig. 4. Parsing rule that contains the same constituents as the error rule in Fig. 3, but differs in constraints

If all comparison operators in the error rule are true, it does not guarantee that this error will be in a final parse tree. For the error rule to succeed, the phrase it covers must be bigger than the phrase for which the parsing rule works. In Fig. 5, the error rule is applied for the three subsequent words, while the parsing rule covers the phrase with five words which include the shorter phrase. Thus, there is no error.

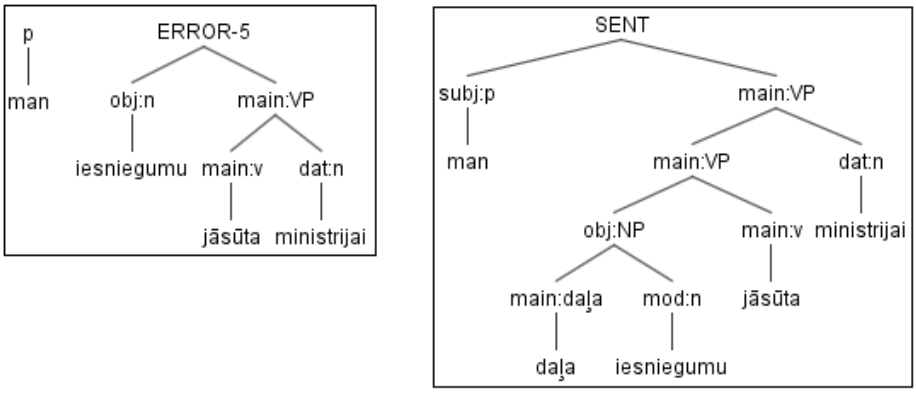

Fig. 5. Parsing example of a grammatically incorrect phrase ‘man iesniegumu jāsūta ministrijai’ (‘I application must send to the ministry’) on the left and a parse tree for a correct phrase ‘man daļa iesniegumu jāsūta ministrijai’ (‘I part of applications must send to the ministry’) containing three subsequent words from the left side phrase

4 Parsing with the Extended CFG Rules

We use the Cocke-Younger-Kasami (CYK) algorithm [26] for parsing. It allows partial parsing which is important for ungrammatical sentences. This algorithm requires grammar in the Chomsky Normal Form. During compilation, our rule compiler expands the rules with optional parts, inserts the unary rules, and transforms the rules into binary form.

Our parser generates parse trees in two formats – either constituency parse trees or dependency parse trees. Every constituency parse tree can be converted to a dependency parse tree by traversing the parse tree from the root node to the child with a syntactic role "main" first and moving it to the parent position. See Fig. 6, for an example of constituency and dependency trees for the same sentence.

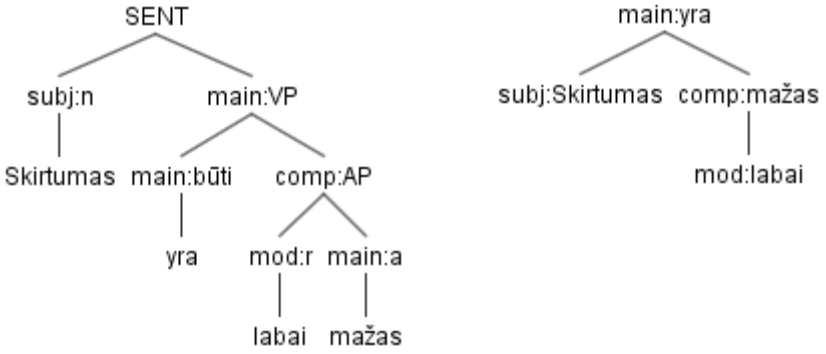

Fig. 6. Constituency (on the left) and dependency (on the right) parse trees for the sentence 'Skirtumas yra labai mažas.' ('The difference is very small.')

5 Evaluation

We have developed several sets of rules for parsing and for grammar checking that are used in different combinations. Correct syntax rules are used to determine the syntactic structure of the sentence. Correct syntax rules, together with error rules, are used to find syntactic errors in a text. Another error rule set is used to find errors in subsequent words in an incorrect text.

Table 2. Rule set statistics

Rule type	Latvian	Lithuanian
Correct syntax rules	580	179
Error rules which depend on phrases described by correct syntax rules	263	72
Error rules which contain only terminal symbols	239	560
Total	**1082**	**811**

We use several data sets to evaluate the quality of the grammar checkers.

The **Latvian Balanced corpus** contains 9,358 sentences. Sentences are taken from different types of texts – news, travel information, student papers, legal texts, blogs, e-mails, non-edited marketing materials, project drafts, etc. They represent the diversity of texts that the potential grammar checker user might check.

The **Lithuanian Balanced corpus** contains 10,000 sentences that are split in two similar parts – each part contains 5,000 sentences. The content is similar to the Latvian balanced corpus.

The **Corpus of Latvian Student papers** contains texts from student essays and abstracts of scientific papers. Intentionally, low quality texts with many grammatical errors are included in this corpus. This corpus is split in two similar parts (development and test) – each part contains 5,157 sentences.

The grammar checker can find a wide variety of errors. All errors that can be flagged by our grammar checking system are divided into 23 groups. This division is based on the theory of language syntax, theoretical literature about common error types in language, and analysis of real texts from different domains.

Simple punctuation errors include errors related to incorrect usage of whitespace characters and punctuation marks for general, language-independent cases (e.g., number of brackets). They are located using search with regular expressions.

Capitalization errors are related to incorrect usage of upper/lower case letters in named entities.

Style errors include different errors for cases where some words are misused, overused, used ungrammatically, or the word sequence is borrowed directly from another language. If the style error rule describes the misusage of individual words, lexical statements must be added to the error rule. If the style error rule describes a phrase with ungrammatically ordered sub-phrases, a set of correct grammar rules together with error rules must be used as in the case of syntax errors.

Syntax errors are related to different agreement errors, punctuation errors in subclauses, wrong mood for a verb, word or sentence part sequence errors, errors in address, punctuation errors in grouping, comma errors (between equal parts of sentence, in insertions, etc.), and other syntax errors. To locate the syntax errors, full parsing of the sentence must be done. A set of correct grammar rules is applied together with the rules describing the errors.

At first, we manually annotated the above mentioned evaluation corpora to create a Gold Standard. During evaluation, the Gold Standard was updated with previously unknown cases and incorrect error detection samples from the output of the grammar checker.

The record in the Gold Standard has four TAB separated fields: sentence number in corpus, error type or symbol ‘0’ for a correct sentence, correctness tag (‘COR’ - correct or ‘INCOR’- incorrect) and suggested correction (Fig. 7).

```
1  ERROR-1 COR ‘Vakar susitikau su geru draugu.‘
2  0
```

Fig. 7. Records in the Gold Standard added by annotator. The first sentence has error of type ERROR-1 and suggestion for error correction, the second sentence is correct.

For negative samples, i.e., if there is no error of type x in the sentence, the '!' symbol appears before the error type, and a phrase for which there should not be this error appears after the INCOR mark.

Fig. 8 shows sentences from the Gold Standard which were previously annotated, and afterwards the information was updated. For the first sentence, the grammar checker detects ERROR-1, but generates the wrong corrections. In the second sentence, the grammar checker incorrectly detects ERROR-1.

```
1  ERROR-1 INCOR 'Vakar susitikau su geru draugai.'
2  !ERROR-1  INCOR 'aš skai iau'
```

Fig. 8. Records appended to the Gold Standard after running grammar checker

In order to evaluate the quality of the grammar checker in general and for certain error types specifically, we calculate recall, precision, and f-measure [21]. Evaluation results are summarized in Table 3.

Table 3. Evaluation results for all error types and for the two most common error types

Corpus	Error type	Precision	Recall	F-measure
Lithuanian Balanced	all error types	0.898	0.412	0.564
	vocabulary errors	0.956	0.535	0.686
	incorrect usage of cases	0.734	0.259	0.383
Latvian Balanced	all error types	0.780	0.455	0.575
	punctuation in sub-clauses	0.757	0.643	0.695
	punctuation in participle clauses	0.617	0.671	0.643
Latvian Student papers (dev)	All error types	0.652	0.231	0.341
	punctuation in sub-clauses	0.706	0.586	0.641
	punctuation in participle clauses	0.656	0.560	0.604
Latvian Student papers (test)	all error types	0.753	0.203	0.320
	punctuation in sub-clauses	0.773	0.588	0.668
	punctuation in participle clauses	0.766	0.685	0.723

We also performed a human evaluation of the grammar checker on 150 sentences (divided into five files containing 30 sentences each) from the Corpus of Latvian student papers. Five human annotators were involved; each file was evaluated by two annotators.

The evaluation was done in two steps – without help from the grammar checker and with help from the grammar checker. At first annotators were asked to find and correct grammar errors in a file. The next day, the files with the same sentences where given to human annotators for correction, but this time files also contained information about how the grammar checker would correct these sentences.

Table 4. Human evaluation results

Situation	Cases %	Hypothesis
First step - annotators agree	62.50	Annotators often disagree. The sentences are not simple, and annotators have different language skills.
Second step - annotators agree	85.83	Agreement is higher, and the grammar checker helps.
Second step – annotator corrects the previously unnoticed error, if the grammar checker suggests it	51.43	The grammar checker helps, but annotators do not blindly accept all suggestions made by the grammar checker.
Second step – annotator does not correct the previously corrected error, if the grammar checker does not suggest it	70.97	Annotators do not read sentences as carefully as before, and they rely on the grammar checker.
First step - sentences which annotator corrects	37.04	
Second step - sentences which annotator corrects	27.78	
Sentences which grammar checker corrects	27.33	

The errors which the human annotators did not notice before, but fixed after seeing the grammar checker's suggestions are: date formatting errors, punctuation errors in participle clauses and sub-clauses, wrong forms of similarly written words, and writing style errors.

Although our main task was to evaluate the grammar checkers, we also did an initial evaluation of the Lithuanian parser. For evaluation, we created a Gold Standard containing 115 correct dependency parse trees. As the syntactic rules that have been developed so far do not cover all of the syntactic constructions used in the Lithuanian language, we included sentences in the Gold Standard from news texts which do not have a very complex structure, but still represent the main syntactic constructions of the Lithuanian language. We compared dependency trees from the Gold Standard with dependency trees generated by the parser. As it is hard to compare two dependency trees, we first converted them into triplets: <parent>:<parent start position in sentence>, < syntactic role>, <child>:<child start position in sentence> (see Fig. 9).

yra:10	subj	Skirtumas:0
yra:10	comp	mažas:20
mažas:20	mod	labai:14

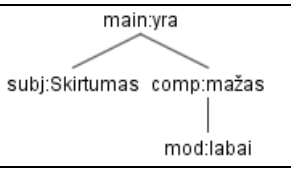

Fig. 9. Triplets for the sentence ‘Skirtumas yra labai mažas’ (‘The difference is very small’)

The quality of the parser is calculated by measuring the precision and the recall of triplets. For the initial Gold Standard, we obtained precision - 0.935 and recall - 0.922.

6 Conclusion and Future Work

In this paper, we introduced extended CFG formalism for grammar checking of inflected languages, allowing powerful grammar checkers to be built for a practical application. The proposed grammar checking formalism has been implemented and tested for Latvian and Lithuanian. The obtained precision and recall numbers (precision over 0.78 and recall over 0.41) allow us to conclude that it can be used in commercial applications.

Our investigations also show that it can also be used for grammar checking of other inflectional languages. We have investigated its possible application to the Polish language. The Polish language belongs to the West-Slavonic group of the Indo-European family of languages. The main verbal morpho-syntactic features (tense, person, aspect, mode, and voice) and nominal features (case, number, and gender) as well as the syntactic structure of the sentence (main parts - subject and predicate; secondary parts - attribute, adverbial modifier, and complement) are similar to the Baltic language group. Our proposed formalism allows the describing of such structures. Similar error types that are common for the Baltic languages are also common in Polish and other Slavic languages: agreement between words, wrong noun case usage, punctuation errors in subclauses, errors in negation, subject and predicate agreement errors, etc.

Our proposed formalism can also be used for named entity recognition and information extraction, and it can be incorporated into hybrid machine translation systems.

The next steps are to add the possibility to specify the weights or probabilities of the rules in the formalism and to implement the CYK algorithm for parsing weighted CFG grammar.

Acknowledgements. The research leading to these results has received funding from the research project “Information and Communication Technology Competence Center” of EU Structural funds, contract nr. L-KC-11-0003 signed between ICT Competence Centre and Investment and Development Agency of Latvia, Research No. 2.8 ”Research of automatic methods for text structural analysis”.

References

1. Arppe, A.: Developing a grammar checker for Swedish. In: 12th Nordic Conference in Computational Linguistics (Nodalida 1999), pp. 13–27. Trondheim (2000)
2. Atwell, E.S.: How to detect grammatical errors in a text without parsing it. In: 3rd Conference of the European Chapter of the Association for Computational Linguistics, pp. 38–45. Association for Computational Linguistics, Copenhagen (1987)
3. Backus, J.W.: The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference. In: International Conference on Information Processing, pp. 125–132. UNESCO (1959)

4. Bates, M.: The theory and practice of augmented transition networks. In: Bolc, L. (ed.) Natural Language Communication with Computers. LNCS, vol. 63, pp. 191–254. Springer, Heidelberg (1978)
5. Birn, J.: Detecting grammar errors with Lingsoft's Swedish grammar checker. In: 12th Nordic Conference in Computational Linguistics (Nodalida 1999), pp. 28–40. Trondheim (2000)
6. Chomsky, N.: Syntactic structures. Mouton, The Hague (1957)
7. Deksne, D., Skadiņš, R.: CFG Based Grammar Checker for Latvian. In: 18th Nordic Conference in Computational Linguistics (NODALIDA 2011), pp. 275–278. Riga (2011)
8. Domeij, R., Knutsson, O., Carlberger, J., Kann, V.: Granska: An efficient hybrid system for Swedish grammar checking. In: 12th Nordic Conference in Computational Linguistics (Nodalida 1999), pp. 49–56. Trondheim (2000)
9. Ehsan, N., Faili, H.: Grammatical and context-sensitive error correction using a statistical machine translation framework. Software: Practice and Experience 43(2), 187–206 (2013)
10. Gazdar, G.: Generalized Phrase Structure Grammar. Harvard University Press (1985)
11. Heidorn, G.E.: Intelligent writing assistance. In: Dale, R., Moisl, H., Somers, H. (eds.) Handbook of Natural Language Processing, ch. 8, pp. 181–207. Marcel Dekker, New York (2000)
12. Hagen, K., Johannessen, J. B., Lane, P.: Some problems related to the development of a grammar checker. Paper presented at NODALIDA 2001, the 2001 Nordic Conference in Computational Linguistics, May 21–22, 2001 (2001)
13. Izumi, E., Uchimoto, K., Saiga, T., Supnithi, T., Isahara, H.: Automatic error detection in the Japanese learners English spoken data. In: 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003), Sapporo, pp. 145–148 (2003)
14. Kaplan, R.M., Bresnan, J.: Lexical-Functional Grammar: A formal system for grammatical representation. In: Bresnan, J. (ed.) The Mental Representation of Grammatical Relations, pp. 173–281. The MIT Press, Cambridge (1982)
15. Karlsson, F.: Constraint Grammar as a framework for parsing running text. In: 13th International Conference on Computational Linguistics (COLING 1990), Helsinki, vol. 3, pp. 168–173 (1990)
16. Martins, R.T., Hasegawa, R., Das Gracas VolpeNunes, M., Montilha, G., De Oliveira, O.N.: Linguistic issues in the development of ReGra: A grammar checker for Brazilian Portuguese. Natural Language Engineering 4(4), 287–307 (1998)
17. Naber, D.: A rule-based style and grammar checker. Master's thesis, University of Bielefeld (2003)
18. Ng, H.T., Wu, S.M., Wu, Y., Hadiwinoto, C., Tetreault, J.: The CoNLL-2013 Shared Task on Grammatical Error Correction. In: 17th Conference on Computational Natural Language Learning (CoNLL 2013), pp. 1–12. Association for Computational Linguistics (2013)
19. Pereira, F., Warren, D.: Definite clause grammars for language analysis–A survey of the formalism and a comparison with augmented transition networks. In: Artificial Intelligence, vol. 13(3), pp. 231–278. (1980)
20. Pollard, C., Sag, I.A.: Head-driven phrase structure grammar. University of Chicago Press, Chicago (1994)
21. Van Rijsbergen, C.J.: Evaluation. In: Information Retrieval, 2nd edn. Butterworth, Newton (1979)
22. Sågvall-Hein, A.: A Chart-Based Framework for Grammar Checking. In: 11th Nordic Conference in Computational Linguistics (Nodalida 1998), pp. 68–80 (1998)

23. Schmidt-Wigger, A.: Grammar and Style Checking in German. In: 2nd International Workshop on Controlled Language Applications (CLAW 1998). Language Technologies Institute, Carnegie Mellon University, Pittsburgh (1998)
24. Shaalan, K.: Arabic Gramcheck: A Grammar Checker for Arabic. Software: Practice and Experience 35(7), 643–665 (2005)
25. Sjöbergh, J., Knutsson, O.: Faking errors to avoid making errors: Very weakly supervised learning for error detection in writing. In: Recent Advances in Natural Language Processing IV (RANLP 2005), Borovets, pp. 506–512 (2004)
26. Younger, D.: Recognition and parsing of context-free languages in time n3. Information and Control 10(2), 189–208 (1967)
27. Vosse, T.: The Word Connection. Grammar-Based Spelling Error Correction in Dutch. Neslia Paniculata, Enschede (1994)
28. Xing, J., Wang, L., Wong, D.F., Chao, S., Zeng, X.: UM-Checker: A Hybrid System for English Grammatical Error Correction. In: 17th Conference on Computational Natural Language Learning (CoNLL-2013), vol. 34 (2013)