# A Grammar Checker for English 

Joakim Simonsson<br>Lund Institute of Technology, Sweden<br>joakim@joakims.com

f02ft@student.lth.se


#### Abstract

This project report is about how a simple but scalable grammar checker for the English language can be implemented. All steps from the plain text to the input for the grammar checker are briefly explained. The ideas and implementation of the grammar checker and its possibilities and limitations are described in more detail. The performance and results of the grammar checker are evaluated and discussed.


## 1 Introduction

The idea to have a computer program check the grammar of a text as you write is as old as the first word processors themselves. The first attempts and implementations trying to handle the problem used large dictionaries of phrases and sentences with examples of grammatically incorrect text. The problem with this approach is that very large dictionaries are needed - one entry is needed for each possible error. Grammatical rules operates on relation between words - not on the word themselves. This means that the number of possible errors increases exponentially with each new word the system can handle. To get better results, more complex rules than just error matching must be used. Thus making an analysis on a deeper level - checking relations between words. This is the way commercially available grammar checkers work.

The goal of this project has been to create a grammar checker that produces results that can be tested and compared with existing systems. A simple
grammar checker that uses manually written rules acting on relation between words has been developed. In the development of the program, scalability and the possibility to easily add more rules have been important factors. The grammar checker handles only two types of rules at the moment, but the system is easily scalable.

The program takes a dependency parsed text as input. To make the dependency parsing, the text first needs to be processed. How this is done is explained in section 2 below. Then a more detailed description of the grammar checker follows in section 3 .

## 2 Text Processing

The grammar checking process consists of several steps. The purpose of these steps is to generate a part of speech tagged text with dependency relations. The text that is going to be grammar checked, hereafter referred as the user text, is tokenized, part of speech tagged, and finally parsed for dependencies.

A bash script that we have made automates these steps. This script takes the user text as an argument.

### 2.1 Text Tokenization

The purpose of tokenizing a text is to split adjoining elements, such as punctuations and verb contractions. The result of this splitting is a text where each element is more precise. For example, the verb contraction "You're" is split into "You" and "'re", one noun and one verb respectively.

We use a script made by Robert MacIntyre to do the tokenization. The script produces Penn Treebank tokenization.

### 2.2 Part of Speech Tagging

To give each word its correct part of speech tag, we use MXPOST. This program takes a tokenized text as input and outputs a part of speech tagged text ${ }^{1}$. MXPOST comes with a pre-trained model that we use.

### 2.3 Dependency Parsing

To create dependency relations between words, we use the MALT parser. We wrote a Perl script that converts the output of MXPOST into a format that is suitable for the MALT parser. We configured the MALT parser so it outputs the dependency relations in an XML format ${ }^{2}$.

As with MXPOST, MALT parser also comes with a pre-trained model. We use this model in our tests.

## 3 Grammar Checker

The user text is now part of speech tagged and dependency parsed. This is what is needed to complete the analysis of the text. Now the relations of the words in the text can be extracted and checked.

The part of this project that evaluates grammatical relations is written in Java. It takes dependency parsed text as input (section 3.2) and evaluates rules written in XML (section 3.3). The XML rules gives an opportunity to easily specify grammatical rules, however there are errors that cannot be formulated in such a way the XML rules are defined (section 4.3). The following sections explains in detail how the grammar checker works.

### 3.1 Basic Idea

The basic idea of the grammar checker is to analyse the relation between two words. Every relation is defined by four possible parameters for each word. These parameters are extracted from the dependency parsed text. The parameters are id, form, postag and deprel. The id parameter specifies at which position in the sentence the word occurs. The form parameter is the actual lexical word. The postag parameter is the part of speech tag of the word, and the deprel parameter is the dependency relation of the current word to its head. A relation is thus

[^0]

Figure 1: An illustration of a sentence object used by the grammar checker
defined by the four parameters of one word, and the four parameters of the head of the first word - giving a total of eight variables for each relation. By defining which relations that are illegal in the XML rules (section 3.3), the program can search for such relations and return an error in such case.

### 3.2 Program

First the XML rules are parsed by the program using a standard Java Open Source XML parser ${ }^{3}$ and a RuleSet object is created that contains the set of rules that should be used in the analysis. The dependency parsed text is then read by the program with the same XML parser and Word and Sentence objects are created (see figure 1). Each Sentence object consist of a number of Word objects and each Word has four variables specifying the four parameters of the actual word. The Word object also has a function returning the words head.

The rules are now applied to one sentence at a time, extracting each word and its head and the relation defined by their parameter values. Each relation is compared with all illegal relations, and if an illegal relation is matched, the error message defined in the rules is returned by the program, and the program continues with the rest of the relations.

[^1]
### 3.3 XML Rules

The XML rules are defined by the syntax given in appendix A.2.1. Each rule consists of a rule tag that has three identifiers: id, name and message that specifies the id, name and error message of the rule. Every rule consists of one ore many illegal tags, by which the illegal relations are defined. The illegal tag has eight identifiers specifying the four parameters of the first word and the four parameters of the first words head thus defining a relation. Two examples of rules are given in appendices A.2.2 and A.2.3 defining one rule for subject verb agreement and one rule for pronoun verb agreement (in present tense).

## 4 Performance

In our system, the time to perform grammar checking on a few sentences (10-20) takes between 5 and 10 minutes: The MALT parser takes between 5 and 10 minutes, other programs in total, takes less than 5 seconds. Clearly, MALT parser is the bottleneck and disqualifies to be part of a real-time grammar checker.

Since integrated grammar checkers in word processors initialize only once, we could omit the time it takes for each program to initialize, and only count the actual execution time, to give a more fair comparison. In this case, the execution time for all programs except the MALT parser would decrease to just a few seconds, but the MALT parser still would take minutes to process its data.

### 4.1 Results

The texts that we used to test the correctness of our grammar checker were both texts copied from the web and texts that we wrote on our own. The texts from the web were mainly copied from online newspapers and Wikipedia. We assumed that these texts were free from grammatical errors.

We injected grammatical errors in the texts and used our system to find these. The idea of the injected errors was to test the rules that we wrote for our grammar checker - subject verb agreement and pronoun verb agreement errors.

In total, we injected 60 errors in the different texts. Our grammar checking system finds 29 of these errors. It also reports one additional error incorrectly,
i.e. one grammatically correct sentence is reported as an error. The correctness of our grammar checker is slightly below $50 \%$.

The reason why our grammar checking system does not detect all of the injected errors is because of that the text is incorrectly part of speeched tagged and dependency parsed. An error in the part of speech tagging often leads to an error in the dependency parsing aswell. To improve the results, a more accurate part of speech tagger (or a different training set for the part of speech tagger) would increase the over all correctness in our grammar checking system.

### 4.2 Comparison with Microsoft Word

We found it interesting to compare our system with the grammar checker in Microsoft Word. We grammar checked the same error injected texts with Word's grammar checker and it found 33 of the 60 errors. This is slightly better. Word found errors that we did not find. However, what we found noteworthy was that our system found errors that Word did not find. See Table 1 for results.

### 4.3 Limitations

All steps in the analysis of the text are limited by the performance of the specific program that handles each step. The part of speech tagger is not perfect, and the errors it makes are inherited by the dependency parser. The dependency parser itself is not perfect and makes additional errors. The XML rules are limited to single relations between two words, and does not handle multiple relations and coherence. All these factors limits the ultimate performance of the grammar checker.

The run time of the system is also unfeasible for a real time application.

## 5 Conclusions and Future Work

In this project we have shown how a working grammar checker can be constructed. Even though the grammar checker only handles two types of errors, the system is easily extendable by writing more rules in the XML format defined in appendix A.2.1. We have also automated the steps necessary from user text to a dependency parsed text ready for analysis. The main limitation in the working system is the execution time for the dependency parsing. This could

| Injected Errors | Word | Our | Comment |
| :---: | :---: | :---: | :--- |
| 2 | 1 | 2 |  |
| 5 | 1 | 1 | Different errors |
| 5 | 1 | 1 | Same error found |
| 12 | 5 | 5 | Our incorrectly finds one error |
| 23 | 17 | 15 |  |
| 13 | 8 | 6 | Different errors |

Table 1: Errors found in texts. Performance comparison with Microsoft Word. The tests have been made on texts with artificially inserted errors.
be improved and optimised by using another dependency parser, but this is not within the scope of this project. More rules can easily be added and evaluated, but our focus has been to create a working system that could be evaluated and compared with other systems on simple test cases.

The limitation of the XML rules could be improved by defining more elaborate rules. However we have shown that descent results can be acquired by using only the limited, single relation rules. In these cases, the system performs competitively compared to Microsoft Word, on finding the errors defined by the XML rules.

## 6 Acknowledgements

The authors wish to thank Pierre Nugues for supervision and support and Johan Hall for e-mail correspondence.

## A Appendix

## A. 1 Text Processing Examples

## A.1.1 Original Text

The dogs are hungry.
The dog is hungry.
I am tired.
I is tired.

## A.1.2 Tokenized and Part of Speech Tagged Text

The_DT dogs_NNS are_VBP hungry_JJ ._.
The_DT dog_NN is_VBZ hungry_JJ •_.
I_PRP am_VBP tired_VBN •_.
I_PRP is_VBZ tired_VBN ._.

## A.1.3 Dependency Parsed Text

```
<sentence id="1" user="MaltParser" date="2007-11-27">
    <word id="1" form="The" postag="DT" head="2" deprel="NMOD" />
    <word id="2" form="dogs" postag="NNS" head="3" deprel="SUB"/>
    <word id="3" form="are" postag="VBP" head="0" deprel="ROOT"/>
    <word id="4" form="hungry" postag="JJ" head="3" deprel="PRD"/>
    <word id="5" form="." postag="." head="3" deprel="VMOD" />
</sentence>
<sentence id="2" user="MaltParser" date="2007-11-27">
    <word id="1" form="The" postag="DT" head="2" deprel="NMOD" />
    <word id="2" form="dog" postag="NN" head="3" deprel="SUB"/>
    <word id="3" form="is" postag="VBZ" head="0" deprel="ROOT" />
    <word id="4" form="hungry" postag="JJ" head="3" deprel="PRD"/>
    <word id="5" form="." postag="." head="3" deprel="VMOD"/>
</sentence>
<sentence id="3" user="MaltParser" date="2007-11-27">
    <word id="1" form="I" postag="PRP" head="2" deprel="SUB"/>
    <word id="2" form="am" postag="VBP" head="0" deprel="ROOT"/>
    <word id="3" form="tired" postag="VBN" head="2" deprel="PRD"/>
    <word id="4" form="." postag="." head="2" deprel="VMOD"/>
</sentence>
<sentence id="4" user="MaltParser" date="2007-11-27">
    <word id="1" form="I" postag="PRP" head="2" deprel="SUB"/>
    <word id="2" form="is" postag="VBZ" head="0" deprel="ROOT"/>
    <word id="3" form="tired" postag="VBN" head="2" deprel="PRD"/>
    <word id="4" form="." postag="." head="2" deprel="VMOD" />
</sentence>
```


## A. 2 XML Rules

## A.2.1 Syntax

<rule id name message>
<illegal id word postag deprel headId headWord headPostag headDeprel></illegal>
<illegal id word postag deprel headId headWord headPostag headDeprel></illegal>
...
</rule>

## A.2.2 Subect Verb Agreement Error

```
<rule id="SVA" name="Subject Verb Agreement"
    message="Possible Subject Verb Agreement Error">
            <illegal postag="NN" headpostag="VBP" deprel="SUB"></illegal>
            <illegal postag="NNS" headpostag="VBZ" deprel="SUB"></illegal>
</rule>
```


## A.2.3 Pronoun Verb Agreement Error

```
<rule id="PVA" name="Pronoun Verb Agreement"
message="Possible Pronoun Agreement Error">
    <illegal word="I" postag="PRP" headpostag="VBZ" deprel="SUB"></illegal>
    <illegal word="you" postag="PRP" headpostag="VBZ" deprel="SUB"></illegal>
    <illegal word="he" postag="PRP" headpostag="VBP" deprel="SUB"></illegal>
    <illegal word="she" postag="PRP" headpostag="VBP" deprel="SUB"></illegal>
    <illegal word="it" postag="PRP" headpostag="VBP" deprel="SUB"></illegal>
    <illegal word="we" postag="PRP" headpostag="VBZ" deprel="SUB"></illegal>
    <illegal word="they" postag="PRP" headpostag="VBZ" deprel="SUB"></illegal>
</rule>
```


## A. 3 Example Text

In the following example, errors found by our grammar checker are bold face.

It is hard to find good examples. So we decided to construct some examples that shows the power of our grammar checker. All students here at LTH drinks lots of coffee. I also wants to eat today, but I has no food left. The tables fall down. The tables falls down. How come Word do not find an easy error like this? Our program find it easy! Problems of this sort are hard to find. But a problem of this sort is hard to find. How many sentences does we need? I does not know. But ask Joakim, he know for sure! I supposes this are a good way to start. Sometimes there is more than one error in a sentence. Just because it seem wrong, it's not definitely wrong. I can come up with many sentences that seems to be wrong at first sight. This is the last sentence I can be bothered to writes. Did you spot the errors?


[^0]:    ${ }^{1}$ See appendix A.1.2 for an example of a part of speech tagged text.
    ${ }^{2}$ See appendix A.1.3 for an example of a dependency parsed text.

[^1]:    ${ }^{3}$ SAX Parser, Xerxec Java Parser 1.4.4, Released under the Apache Software License

