h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

HÁSKÓLINN Í REYKJAVÍK
REYKJAVÍK UNIVERSITY

Hochschule Darmstadt & Reykjavík University
Departments of Computer Science

LISGrammarChecker: Language Independent Statistical Grammar Checking

Master Thesis to achieve the academic degree
Master of Science (M.Sc.)

Verena Henrich and Timo Reuter
February 2009

First advisor: Prof. Dr. Bettina Harriehausen-Mühlbauer
Second advisor: Hrafn Loftsson, Ph.D., Assistant Professor

Master Thesis

"There is something funny about me grammar checking a paper about grammar checking..."

William Scott Harvey

Abstract

People produce texts, and therefore the use of computers rises more and more. The grammatical correctness is often very important and thus grammar checkers are applied. Most nowadays grammar checkers are based on rules, but often they do not work as properly as the users want. To counteract this problem, new approaches use statistical data instead of rules as a basis. This work introduces such a grammar checker: LISGrammarChecker, a Language Independent Statistical Grammar Checker.

This work hypothesizes that it is possible to check grammar up to a certain extent by only using statistical data. The approach should facilitate grammar checking even in those languages where rule-based grammar checking is an insufficient solution, e.g. because the language is so complex that a mapping of all grammatical features to a set of rules is not possible.

LISGrammarChecker extracts n-grams from correct sentences to built up a statistical database in a training phase. This data is used to find errors and propose error corrections. It contains bi-, tri-, quad- and pentagrams of tokens and bi-, tri-, quad- and pentagrams of part-of-speech tags. To detect errors every sentence is analyzed with regard to its n-grams. These n-grams are compared to those in the database. If an n-gram is not found in the database, it is assumed to be incorrect. For every incorrect n-gram an error point depending on the type of n-gram is assigned.

Evaluation results prove that this approach works for different languages although the accuracy of the grammar checking varies. Reasons are due to differences in the morphological richness of the languages. The reliability of the statistical data is very important, i.e. it is mandatory to provide enough data in good quality to find all grammatical errors. The more tags the used tagset contains, the more grammatical features can be represented. Thus the quality of the statistical data and the used tagset influence the quality of the grammar checking result. The statistical data, i.e. the n-grams of tokens, can be extended by n-grams from the Internet. In spite of all improvements there are still many issues in finding reliably all grammatical errors. We counteract this problem by a combination of the statistical approach with selected language dependent rules.

Contents

I. Introduction

II. Statistical Grammar Checking

III. Evaluation

List of Figures

List of Tables

Part I.

Introduction

1 Introduction

Nowadays people expect their computer systems to support a lot of functionality, one such functionality includes writing documents and texts. It is important in many domains to produce texts which are correct with regard to their syntax and grammar. This is no easy task. People who write texts in foreign languages are unsure about correct syntax and grammar. The demand for computer programs which help to produce texts with high quality, i.e. a text with correct grammar and good style, increases.

Almost everyone writing a document on a computer uses at least one tool to check, for instance, the spelling of the words. Spell checking can be done simply with a dictionary and few rules. Today, all important word processors, for example OpenOffice [Ope] and Microsoft Word [Micb], provide a spell checker in their standard configuration. Spell checking works quite well for many languages. This saves lots of time and is a first step towards a high quality text.

Also, since some time, grammar checking tools have arisen in popularity, by companies like Microsoft that have introduced grammar checker tools to their Office Suite. In comparison to spell checking, grammar checking is much more complex and has thus left behind feature completion and correctness. In theory, grammar checkers should work like spell checkers already do; but, current grammar checkers reveal several limitations. It is frustrating for

the user to still see wrong phrases that have not been marked as wrong and, even more exasperating, correct phrases marked as incorrect.

Thus, in spite of the help of a computer program, manual reviewing is indispensable in order to get high quality texts.

1.1. Motivation

Grammar checking is important for various aspects. It improves the quality of text, saves time while writing texts, and supports the learning of a language. There are several grammar checking approaches. The most common method of grammar checking is rule-based. These grammar checkers are established to some degree. They work quite well for chosen languages. Rule-based grammar checkers work with a set of rules which represent the grammatical structure of a specific language. This means that a rule-based grammar checker is language-dependent. The rules can cover almost all features of a language which makes the approach powerful. The more features are implemented, the better are the results but the more complex the systems. If a language is morphologically rich or if a lot of different phrase structures are possible, the complexity of a grammar checker quickly increases. All languages differ in their grammatical structure, which makes it nearly impossible to support more than one language with the same rule set. A rule-based grammar checker can be considered as static—once written for a specific language it depends on and supports only this language.

This means that a rule-based grammar checker needs to be written separately for each language that need to be supported. This is a very time- and resource-consuming task. Furthermore, some languages are so complex that a mapping of all grammatical features to a set of rules is not possible. It is not even possible to write a rule-based grammar checker for every natural language. Although a rule-based approach seems to be very promising, even today a feature-complete grammar checker is not available—not even for a morphologically simple language like English.

A study over ten years [Kie08] shows, that all famous grammar checkers are far from being perfect. This evaluation compares several rule-based grammar checkers, like the grammar checker from Microsoft Word [Micb] and the LanguageTool [Lanb] from OpenOffice.org [Ope].

A poll in the Seattle Post-Intelligencer article about the usefulness of the Microsoft Word grammar checker [Bis05] shows that the majority are of the opinion that a grammar checker is not as useful as it should be. Even if this poll does not represent all users of the Microsoft Word grammar checker, our conclusion is that the need for accurate grammar checkers is high.

New fields of grammar checking arise through more power, storage capabilities and speed of today's computer systems. Statistical data is freely available through the Internet, e.g. through search engines, online newspapers, digital texts, and papers. These basic principles lead to a new idea about grammar checking, which leaves the area of rules, and steps into a statistical approach.

We are thinking about a simple approach to get a maximum impact. Our statistical approach overcomes the problem of language-dependence. The basic concept is language-independent which can be easily adapted to new languages. It can serve several languages at once, and even benefits those languages whose rules are impossible or not valuable to implement because the language is not widespread enough.

1.2. Goal and Definition

Our goal is to write a language independent grammar checker which is based on statistics. From now on, we call it LISGrammarChecker, which stands for Language Independent Statistical Grammar Checker. In our approach, we do not use rules to identify errors in a text. We use statistical data instead. Language independence is supported, because of the lack of language-dependent rules. Statistical data can be given in every natural language and thus language independence is provided. LISGrammarChecker needs statistical data as a basis to learn what is correct to mark errors, i.e. if a sentence or part of it is not known as correct. The statistical data is the basis for all grammar checking. To get a database that is as complete as possible, we consider the possibility of gaining these data directly from the Internet, for example, through a Google [Gooa] search. Errors are based on wrong grammatical usage, which is assumed if the statistical basis does not know a specific grammatical structure.

A problem, which will probably arise, are misleadingly marked errors, so called false positives. The goal is to keep these false positives as few as possible. We want to counteract this with a concept of thresholds, i.e. we do not intend to mark an error immediately but collect a reasonable amount of error assumptions to state the existence of an error. Along with the detection of an error comes the correction proposal. We want to gain these proposals from the most likely alternatives out of the statistical data.

We start with feeding LISGrammarChecker with English and German data, and evaluate these languages as a proof of concept for the language-independence of the program.

Hypothesis

We assume that it is possible to check grammar up to a certain extent by only using statistical data. This can be done independent from a natural language.

1.3. Structure of this Document

To facilitate a better understanding of this document, the fundamentals are explained in chapter 2. This chapter starts with a definition of the term grammar in a natural language, and explains what we mean with grammar checking and n-grams. We specify a corpus in general and introduce well-known corpora for several natural languages. Furthermore, part-of-speech tagging is introduced, together with tags, tagsets, taggers, and combined tagging.

Chapter 3 focuses on related work in grammar checking. For the two main approaches in grammar checking—rule-based and statistical—state-of-the-art work is presented. We introduce the idea of our language independent statistical grammar checking approach and compare it to existing approaches. In doing so, we point out especially the differences between LISGrammarChecker and existing grammar checkers.

We come to the requirements analysis of our grammar checker in chapter 4. We start presenting our idea in more detail and develop the requirements for LISGrammarChecker. According to these requirements, we analyze the consequences for our work and start to specify what we need to consider. We analyze what would fit best to fulfill our requirements. We present which programming language we use, which corpora we prefer and how we gather our statistical data. We regard the aspects of tagging within our grammar checker. Furthermore, we analyze how to process and store data in an appropriate manner.

The design of LISGrammarChecker is described in chapter 5. We show all components of the system and how these work together to fulfill the requirements from chapter 4. All aspects that are already mentioned are considered to trigger a solution for implementation.

In Chapter 6, where we present the implementation of LISGrammarChecker, we describe all implemented functionality of LISGrammarChecker and the way how we have realized it.

To test the functionality of our approach, we create several test cases in chapter 7. These tests should show how LISGrammarChecker works and reveal problems. Therefore, we train different statistical data corpora and check erroneous sentences.

We evaluate and interpret the results of the test cases in chapter 8. Furthermore, we regard LISGrammarChecker with respect to its functionality and the established requirements. We analyze problems that occurred with the implementation of the program and propose solutions for several aspects that could be done alternatively.

In chapter 9, we conclude all what we have learned from our work as well as show the useful knowledge this approach has to build a language independent statistical grammar checker. Finally we present possible future work to our language independent statistical grammar checking approach in chapter 10.

2 Fundamentals

This section introduces the fundamentals which are necessary to understand basic parts of this work. First, we explain our understanding for natural languages including the term grammar and what grammar checking means for this work. Then, collections of text, i.e. corpora, are explained. Finally, we introduce part-of-speech (PoS) tagging with respect to different types of taggers and combined tagging.

2.1. Natural Languages and Grammar Checking

This section starts with a definition of a grammar in a natural language. An explanation of tokenization follows. We introduce the terms grammar checking and n-grams.

2.1.1. Definition: The Grammar of a Natural Language

We understand a natural language as a way for humans to communicate, including all written and spoken words and sentences. The syntax of a language describes how words can be combined to form sentences. The morphology of a language regards the modification of

words with respect to time, number, and gender. We define a grammar of a natural language as a set of combinations (syntax) and modifications (morphology) of components, e.g. words, of the language to form sentences. This definition follows Rob Batstones explanations in [Bat94].

Grammar of a Natural Language

A grammar of a natural language is a set of combinations (syntax) and modifications (morphology) of components and words of the language to form sentences.

The grammar—syntax and morphology—differ in each language. Conversely, this means that, if there is a difference in syntax or morphology in two languages, these languages are not the same—a language can be distinguished from another by its grammar. An example of two very similar languages is American and British English. Most people would not distinguish these as different languages, but if there is at least one difference in grammar, these languages need to be considered separately, following our definition.

2.1.2. Tokenization

Every item in a sentence, i.e. every word, every number, every punctuation, every abbreviation, etc., is a token. We use the term lexeme as a synonym without any differences.

Tokens & Lexemes

A token is every atomic item in a sentence, e.g. words, numbers, punctuation, or abbreviations. Lexeme is used synonymously.

One challenging part when using text is the task of word and sentence segmentation. This is also known as tokenization. Tokenization is the breaking down of text to meaningful parts like words and punctuation, i.e. tokens. This work is usually done by a tokenizer. Tokenization is a necessary step before tagging (section 2.3) can be done.

There are different ways how to tokenize a text. For example, clitics as can't can be interpreted as one token alone, or it can be split and interpreted as the two words can not. There are several such cases, where more than one possibility can make sense. Another such example is the handling of numbers. The format of numbers can vary greatly, e.g. large numbers can be written without any punctuation as 1125000. But in many languages, it is common to

facilitate the reading for readers and thus numbers are written e.g. with periods as delimiter as in 1.125.000 in a lot of Central European languages or with commas 1,125,000 as in the English language.

Tokenization and Tokenizer

Tokenization is the breaking down of text to meaningful parts like words and punctuation, i.e. the segmentation of text into tokens. The process of tokenization is done by a tokenizer.

The main challenge of tokenization is the detection of the sentence and word boundaries. Some symbolic characters, e.g. ". ? !", maybe used as sentence boundary markers in languages which use the Latin or Cyrillic alphabet. As one can see in the large-number example above, these symbolic characters can have different roles. In the English language the period can be used for several tasks. It is used to mark the end of a sentence, as a decimal delimiter, or to mark abbreviations. This causes problems in determining the correct sentence end marker. To solve this problem a list with abbreviations can be used to distinguish abbreviations and the sentence end marker. But if we consider a sentence with an abbreviation at the end of a sentence, there is an ambiguity which need to be resolved.

Another challenge is the detection of word boundaries. Usually words are delimited by white space characters. There are some exceptions, e.g. tagging (see section 2.3) of multiword expressions, which need a different treatment. An example for this is the expression "to kick the bucket". If the space character is used as the word delimiter, the result is four individual tokens that are tagged individually. Because of its sense "to die", the whole expression needs to be tagged with one tag, here a verb, instead of four individual ones.

This is different in other languages using logographic symbols representing words like Chinese. While Chinese and Japanese use the period as a sentence delimiter, words are not necessarily delimited by white space. One can see here that tokenization is not an easy task and has potential influence on the accuracy of any method, like tagging, which depends on this.

2.1.3. Grammar Checking

The grammar of a natural language describes its syntax and morphology, as explained above. Hence, grammar checking can be described as the verification of syntax and morphology according to the used language. The goal is to check a sentence or text for its grammatical correctness. Tools performing this work are called grammar checkers.

Grammar Checking and Grammar Checker

The verification of syntax and morphology is called grammar checking. The process of grammar checking is done by a grammar checker.

In order to check the grammar of a text, different approaches are used.

Pattern matching A very primitive way is pattern matching. This method works by using a data storage where common grammatical mistakes are stored together with their corrections. A sentence, or part of it, is checked by matching it to some error entry in the data storage. In case of a match, an error is detected, which can then be corrected with the stored correction. This method is quite effective for the patterns that are in the data storage. But there is a lack of generality. Every small difference in grammar needs a new pattern, e.g. the (wrong) sentences "He sell." and "He tell.". When using pattern matching there needs to be two entries in the look up table to correct both errors, even if they differ only in one character—the missing s for third person singular.

Rule-based approach A more common way to do grammar checking is based on rules. Simple rules can be used to detect errors which are very easy to find, e.g. doubled punctuation. If the sentences are more complex, the rules to detect errors become more complicated. If we again take the missing "s" example from above, we can tackle the problem of two entries by defining one rule. This rule would define that the personal pronoun he can only be followed by a third person singular verb and not by the infinitive. If the word he is found, followed by a verb which is not in third person singular, an errors is marked. In this approach, a sentence is parsed in such a way that it matches a certain grammar.

Statistical approach A third method for checking grammar is the statistical approach. The main assumption in this approach is that text can be corrected by only using large amounts of text. These texts form a statistical database which is used to detect errors. When using statistics, two different ways can be used to achieve the goal of correcting grammar. One uses the data directly to compare it with the text which should be corrected. Another one derives a grammar from statistical information which can then be used to check and parse the text.

2.1.4. Types of Grammatical Errors

When people are writing, they make mistakes. In this section we take a look at the most frequent error types in English. Table 2.1 shows chosen errors from Kantz and Yates [KY94].

We do not include all errors from the source. Errors that only concern spelling are not listed. The errors in Table 2.1 are sorted by the mean irritation score. This score describes the average degree of how sensible people respond to an error. This means, if an error is very noticeable, then the mean irritation score is high. The table is ordered from most to least bothersome.

Table 2.1.: Example errors in English [KY94]

Category	Example
you're/your	So **your** off on holiday this week and you haven't had a moment to think about the paperbacks.
their/there	Two principal shareholders wanted to sell **there** combined 42% stake.
sentence fragment	They want to know **as to** why they aren't sleeping, why they want to throw up when they eat.
subject-verb agreement	She and Lorin **is** more than willing to lend if they can find worthy borrowers.
wrong preposition in verb phrase	Then we could **of** celebrate the new year on an agreed first day of Spring.
too/to	The chancellor has been at the scene of **to** many accidents.
were/where	One area **were** the Gulf states do seem united is in their changed relations with outside states.
pronoun agreement	Mr. Hodel also raised concerns that the U.S. might commit **themselves** to an ineffective international treaty.
object pronouns as subjects	And she said Tuesday she was not sure how **her** would vote.
run-on sentences	The shares have fallen this **far they** seldom come back.
tense shift	She looked right at me and she **smiles** broadly.
it's/its	The company said **it's** rate of sales growth for the quarter has slowed from the 33% pace during the second quarter.
lose/loose	I promise if somebody starts playing fast and **lose** with the university, they'll have to answer.
dangling modifier	After reading the original study, **the article** remains unconvincing.

Table 2.1.: Example errors in English [KY94] (continued)

Category	Example
comma splice	It is nearly half past **five, we** cannot reach town before dark.
affect/effect	One not entirely accidental side **affect** of the current crackdown will be a dampening of the merger and acquisition boom.
then/than	I never worked harder in my life **then** in the last couple of years.

2.1.5. Definition: n-grams

Usually, n-grams are a subsequence of neighbored tokens in a sentence, where n defines the number of tokens. For example, "modern houses are" is a trigram, consisting of three neighbored words, as you can see in Figure 2.1. We do not differentiate between the types of token, i.e. if the token is a word, a number, an abbreviation, or a punctuation as punctuation marks, commas, etc. Our understanding of n-grams includes all kinds of tokens. An example is shown in Figure 2.1, where "very secure ." also constitutes a trigram.

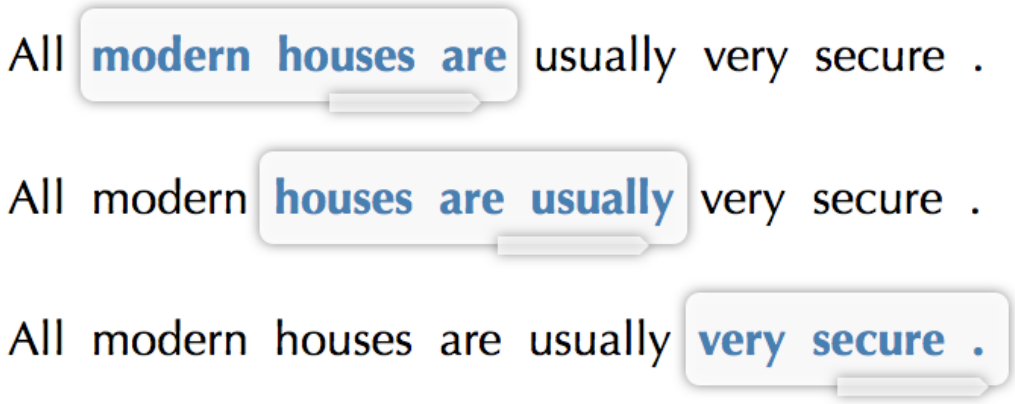

Figure 2.1.: Three example trigrams

A sentence has k-(n-1) n-grams, where k is the sentence length including punctuation, i.e. the number of tokens, and n specifies the n-gram, i.e. how many neighbored tokens. This means, that the example sentence has six trigrams: k-(n-1) = 8-(3-1) = 6, where k = 8, n = 3.

Amount of n-grams

The amount of n-grams in a sentence is k-(n-1), where k specifies the number of tokens in a sentence and n specifies the number of tokens classifying the n-gram.

$$k-(n-1) = \sum \text{n-grams in a sentence}$$

We go one step further and expand the definition of n-grams insofar as we introduce a second n-gram variant. This second variant consists of the tags (see section 2.3) of the tokens and not the tokens (e.g. words) themselves. To distinguish the two n-gram variants, we call them n-grams of tokens and n-grams of tags. For example, we have two trigrams in parallel for "modern houses are":

1. The three consecutive tokens "modern houses are" form a trigram of tokens, and
2. the three tags (here word classes) "adjective noun verb" of these three tokens constitute a trigram of tags.

For the amount of n-grams of tags in a sentence, the same calculation as above applies for the amount of n-grams of tokens: k-(n-1).

n-grams of Tokens & n-grams of Tags

An n-gram of tokens is a subsequence of neighbored tokens in a sentence, where n defines the number of tokens. An n-gram of tags is a sequence of tags that describes such a subsequence of neighbored tokens in a sentence, where n defines the number of tokens.

2.1.6. Multiword Expressions

The term of multiword expressions is already mentioned in the tokenization section. A multiword expression is the combination of at least two lexemes. The meanings of these words differ from their original meaning. For example, the term "White House" does not express a house with white painting. There are different types of multiword expressions. [DMS00]

Multiword entries The example "to kick the bucket" is a typical multiword entry. This means, in case of a rule-based approach, the multiword is not defined by a rule, but has its own entry in the database.

Captoids These are multiword expressions with all words capitalized, e.g. a title as "Language Independent Statistical Grammar Checker". This must not be confused with the capitalization of nouns in German.

Factoids Another subset of multiwords are factoids. These are for example dates or places. Their characteristic is that they can easily be described by rules.

2.1.7. Sphere of Words

Most words influence other words, especially neighbored ones. If we consider the sentence "The boys, who live in this house, are playing with the ball.", the noun boys influences the verb are. This plural form of to be is correct, because of the plural noun boys. The sphere of the single word boys reaches to the distant word are. The correctness can be described as the agreement of the two words.

We want to give two more examples for these types of agreements. The first is the agreement between adverb and verb. This feature is important in English. Let us consider the sentence "Yesterday, the man stayed at home.", containing the temporal adverb yesterday. Whenever this temporal adverb is used in English, the verb must have a certain tense; in this case, simple past. Quite a lot of temporal adverbs force the verb to a fixed tense to assure valid grammar.

Another grammatical dependency exists between adjectives and nouns. Both must agree in several ways, depending on the language. In Icelandic and in German they need to agree in number, case and gender. For example "ein grünes Haus" (singular, neuter, nominative), "ein grüner Baum" (singular, masculine, nominative), "zwei grüne Häuser/Bäume" (plural, masculine/neuter, nominative), and "des grünen Hauses" (singular, neuter, genitive) show different adaptions of the adjectives that are influenced by the characteristics of the nouns. For adjectives in English, there is no distinction for number, gender or case. It makes no difference if we say "the green house" (singular) or "the green houses" (plural). The only exceptions are the two demonstrative adjectives this and that with their plural forms these and those. All other adjectives have only one possible form in modern English.

2.1.8. Language Specialities

As the above examples show, natural languages differ in their morphological features. The complexity of a language depends on the adaption of characteristics, for example, number,

gender and case. Icelandic and German are far more complex than English. They concern different options for all features. Icelandic and German are so called morphologically rich languages. The grammar of a language dictates, if two words agree in their characteristics, e.g. in number, gender and case. In the English language, the differences in case and gender are rare. Thus it is much more simple.

Further differences in natural languages are the lengths of word and sentences. In German, long sentences are common, which is usually not good style in English. Another aspect is the lengths of words. In languages like German and Icelandic, they can be arbitrarily long. This applies for the German compound "Donaudampfschiffahrtsgesellschaft", where 5 nouns are concatenated.

2.2. Corpora—Collections of Text

In this section we give a definition for the term corpus (plural: corpora) and give examples for well-known corpora in chosen natural languages.

2.2.1. Definition: Corpus

We have chosen a definition of a corpus according to that from Lothar Lemnitzer and Heike Zinsmeister in [LZ06]:

Corpus

A corpus is a collection of written or spoken phrases that correspond to a specific natural language. The data of the corpus are typically digital, i.e. it is saved on computers and machine-readable. The corpus consists of the following components:

- The text data itself,
- Possibly meta data which describe the text data,
- And linguistic annotations related to the text data.

2.2.2. Sample Corpora

Every natural language has a distinct set of words and phrases. There exists different corpora for many different languages. We introduce some of the most well-known. The most corpora that we show are English, but we also list German and Icelandic ones. Furthermore, we regard two special types of corpora, Google n-grams and an error corpus.

American National Corpus The American National Corpus (ANC) [IS06] currently contains over 20 million words of American English and is available from the Linguistic Data Consortium [Linb]. The work is currently in progress. Its final version will contain at least 100 million words.

British National Corpus The British National Corpus (BNC) is an analogy to ANC for British English. This is a collection of 100 million words released in its last version in 2007. BNC is already complete. According to the the official BNC website [Bur07] this corpus was built from a wide range of sources. It includes both written and spoken sources and claims to represent a wide selection from 20th century British English.

Brown Corpus An alternative to ANC and BNC is the Standard Corpus of Present-Day American English (also known as Brown Corpus) [FK64]. It consists of approximately 1 million words of running text of edited English prose printed in the United States during the calendar year 1961. Six versions of the Corpus are available. All contain the same basic text, but they differ in typography and format.

Wortschatz Universität Leipzig Another interesting source for corpora is Wortschatz at Universität Leipzig [QH06]. There are corpora available for 18 different languages, e.g. English, German and Icelandic. Their source are newspapers and randomly collected text from the Internet. The German corpus has a size of 30 million sentences, the one for English has 10 million sentences, and the Icelandic corpus consists of 1 million sentences. They are available via a web service. Some of them are partly available for download. Due to copyright issues they are not as big as the ones available online. All texts are split into sentences and stored line by line. The corpora are provided in different sizes. For English the biggest consists of 1 million sentences, for German it contains 3 million sentences with an average of 15 to 16 words per sentence. For Icelandic no downloadable corpus is available.

NEGRA corpus Another well-known German corpus is the NEGRA corpus [BHK+97]. It consists of 355,096 tokens (about 20 thousand sentences) of German newspaper text which is taken from the Frankfurter Rundschau. The corpus is tagged with part-of-speech and annotated with syntactic structures.

Icelandic Frequency Dictionary corpus The common Icelandic corpus is created with the Icelandic Frequency Dictionary [PMB91]. It is published by the Institute of Lexicography in Reykjavik and is considered as carefully balanced. The corpus consists of 500,000 words from texts published in the 1990s. It includes text from five different categories, Icelandic fiction, translated fiction, biographies and memoirs, non-fiction as well as books for children. [Hol04]

Google n-grams This corpus differs from the others insofar as it contains word n-grams and not plain text. Google n-grams [BF06] are a collection of English word n-grams from publicity accessible web pages contributed by Google [Gooa]. They are also available from the Linguistic Data Consortium [Linb]. The n-gram lengths are unigrams up to pentagrams. For every n-gram its observed frequency count is included. Altogether, the corpus includes more than 1 trillion tokens together with 1 billion pentagrams (these are the pentagrams that appear at least 40 times) and 13 million unique words (these are the words that appear at least 200 times). [BF06]

Error corpus An error corpus is a text with intentionally wrong sentences. An example corpus, which contains 1000 wrong English sentences (about 20 thousand words) and their correct counterparts, can be found in [Fos04]. The sources for those error sentences are multifaceted. It *"consists of academic material, emails, newspapers and magazines, websites and discussion forums, drafts of own writing, student assignments, technical manuals, novels, lecture handouts, album sleevenotes, letters, text messages, teletext and signs"* [Fos04].

2.3. Part-of-Speech Tagging

The main task in part-of-speech (PoS) tagging is to assign the appropriate word class and morphosyntactic features to each token in a text. The result is an annotated (or tagged) text, which means that all tokens in the text are annotated with morphosyntactic features. This is useful for several preprocessing steps in natural language processing (NLP) applications, i.e. in parsing, grammar checking, information extraction, and machine translation. Words are not definite in their PoS. Ambiguity can occur through multiple possible morphosyntactic features for one word. Thus still today there are many problems to be solved for reaching good solutions although part-of-speech systems are trained with large corpora.

The process of assigning the word class and morphosyntactic features to words is called tagging. Therefore, the morphosyntactic features are represented as strings, which are denoted as tags. The set of all these tags for a specific natural language constitute a tagset. There exist several tagsets, see section 2.3.1. The process of tagging is performed by a tagger, whose main function is to remove the ambiguity resulted from the multiple possible features for a

word. There exist several methods to perform this task; see section 2.3.2 for an overview of different tagger types.

PoS Tagging and PoS Tagger

Part-of-speech tagging describes the assignment of the appropriate word class and morphosyntactic features to each token in a text. The process of tagging is performed by a tagger.

In the process, the tagging accuracy is measured as the number of correctly tagged tokens divided by the total number of tokens in a text. In general, there can be taggers for all languages, the tagger simply needs to support a language. That there are more taggers for the English language than e.g. for Icelandic is obvious and results from the amount of people speaking a language but also from the morphological complexity. The tagging accuracy obtained for morphologically complex languages is significantly lower than the accuracy obtained for English. There is a possibility to increase the tagging accuracy with combined tagging, which is explained in section 2.3.3.

Tagging Accuracy

The tagging accuracy is measured as the number of correctly tagged tokens divided by the total number of tokens in a text.

2.3.1. Tagset

A tagset subsumes a set of possible tags. There exist several tagsets, which are usually bound to a specific language.

Tags and Tagset

The morphosyntactic features that are assigned to tokens while tagging are represented as strings. These strings are denoted as tags. The set of all tags for one language constitute a tagset.

Penn Treebank tagset One of the most important tagsets for the English language is built by the Penn Treebank Project [MSM93]. The tagset contains 36 part-of-speech tags and 12 tags for punctuation and currency symbols. The Penn Treebank Project is located at the University of Pennsylvania. All data produced by the Treebank is released through the Linguistic Data Consortium [Linb].

Stuttgart-Tübingen Tagset A well known German tagset is the Stuttgart-Tübingen Tagset (STTS) [STST99]. It consists of 54 tags altogether, which are hierarchically structured. 48 of these are part-of-speech tags, and six tags describe additional language parts as e.g. punctuation or foreign language parts. STTS results from the two PoS tagsets that were developed by the Institute for Natural Language Processing, University of Stuttgart, and Department of Linguistics, University of Tübingen.

Icelandic Frequency Dictionary corpus tagset Compared to a tagset in a morphological rich language, the Penn Treebank tagset and the STTS contain only a small amount of tags. A tagset for a morphologically rich language can be much larger, e.g. the main tagset for Icelandic contains about 700 tags. This tagset is constructed through the Icelandic Frequency Dictionary corpus [PMB91]. The tags are not simply tags that describe one morphological feature. Instead, each character in a tag describes a different feature. The first character, for example, denotes the word class. Depending on the word class, there can follow a predefined number and set of characters denoting additional morphological features. For a noun, there can follow features like gender, number and case. For adjectives, degree and declension can follow. Verbs can have a voice, a mood and a tense. [Lof07]

Brown tagset and Münsteraner tagset As said above, a tagset is usually bound to one specific language, but this does not allow the contrary statement that a language can have only one tagset. For English and German there exist at least two. Additionally to the Penn Treebank tagset and the STTS tagset, which both can be seen as the standard tagsets for their targeted language, there exist the Brown tagset [FK64] and the Münsteraner tagset [Ste03]. Both tagsets contain more tags than the Penn Treebank and STTS tagset. The Münsteraner tagset is built similar to the Icelandic one. It also uses each character for a different feature of the word.

Some tagsets define how texts should be tokenized before they are passed to a tagger. For example for the Penn Treebank tagset, the tokenization is proposed in [Pen].

2.3.2. Types of PoS Taggers

There are several methods how a part-of-speech tagger work, e.g. based on rules [Lof08], hidden Markov models [Bra00], probabilistic using decision trees [Sch94], error-driven trans-

formation-based learning [Bri94], or maximum entropy [TM00]. We introduce some of them.

Brill tagger One part-of-speech tagger was introduced by Brill in 1994 [Bri94]. Brill tagger is error-driven and based on transformations. It achieves a relatively high accuracy. The method assigns a tag to each word and uses a set of predefined rules, which is changed iteratively. If a word is known, the most frequent tag is assigned. If a word is unknown, the tag noun is naively assigned. This procedure to change the incorrect tags is applied several times to get a final result.

TnT An example for a tagger based on the implementation of the Viterbi algorithm using hidden Markov models is TnT—Trigrams'n'Tags [Bra00]. This tagger is language independent; new languages can be trained. The main paradigm used for smoothing is linear interpolation. The respective weights are determined by deleted interpolation. Unknown words are handled by a suffix tree and successive abstraction.

Stanford Tagger [TM00] is based on maximum entropy. It uses tagged text to learn a loglinear conditional probability model. This model assigns probabilities to each tag, which are used to evaluate the results. Stanford Tagger is also language independent. It was developed at the Natural Language Processing Group, University of Stanford.

TreeTagger [Sch94] is a probabilistic language independent tagger. It is a tool for annotating text with part-of-speech and lemma information which has been developed within the TC project at the Institute for Computational Linguistics, University of Stuttgart. The TreeTagger has been successfully used to tag several languages, including German and English. It is easily adaptable to other languages, if a lexicon and a manually tagged training corpus are available.

IceTagger Another tagger type is based on rules. These taggers depend mostly on the natural language itself and are thus usually for one specific language with its quirks. An example for an Icelandic language tagger is IceTagger [Lof08]. It is based on a set of rules for the Icelandic language and therefore not suitable for any other languages.

2.3.3. Combined Tagging

Combined tagging is done by a combined tagger which uses the output of two or more individual taggers and combines these in a certain manner to get exactly one tag for each token as the result. It has been shown, for various languages, that combined tagging usually obtains higher accuracy than the application of just a single tagger ([HZD01], [Sjö03]). The reason is that different taggers tend to produce different (complementary) errors and the differences can be exploited to yield better results. When building combined taggers it is thus important to use taggers based on different methods (see section 2.3.2).

Combined Tagging

The combination of more than one single part-of-speech tagger is called combined tagging.

There are several strategies of how the different tagger outputs can be combined. These strategies are denoted as combination algorithms.

Simple voting The simplest combination algorithm represents a majority vote. All individual tagger votes are equally valued while voting for a tag. The sum of all tagger votes are summed up, and the tag with the highest number of votes represents the combined tagging result. In the case of a tie, the tag proposed by the most accurate tagger(s) can be selected. [HZD01]

Weighted voting is similar to simple voting, but gives every tagger output a different weight. For example the taggers which are known to produce a high overall accuracy gets more weight when voting. The weighted votes are also summed up and the tag with the highest result wins. In case of a tie, which is usually rare using this algorithm, the same procedure as stated before applies. [HZD01]

For an overview of further combination algorithms see [HZD01].

3 Related Work

In this section we discuss related work to our grammar checker. As described earlier, there are two main types of grammar checkers. We discuss both methods—rule-based and statistical—and show what has been done so far and how the different approaches work. As examples for the rule-based approach, we explain the system used by the Microsoft Word 97 [Micb] grammar checker and LanguageTool [Lanb], an open source project used as a grammar checker in OpenOffice.org [Ope]. The statistical approach refers to two research works, but none of them are already implemented in a productive used tool.

3.1. Rule-based Approaches

Very well-known grammar checkers are those of Microsoft Word, WordPerfect [Cor], as well as LanguageTool and Grammarian Pro X [Lina]. They are available for many different languages. All of them use a certain amount of rules against which they check sentences or phrases. Thus, a main disadvantage is the limitation to a specific language. Rules can rarely be reused for more than one language. We show this functionality in more detail for the grammar checker in Microsoft Word 97 and the open source tool LanguageTool which can be integrated in OpenOffice.

3.1.1. Microsoft Word 97 Grammar Checker

Microsoft [Mica] splits the grammar checking process in Microsoft Word 97 [Micb] into multiple stages, as we can see in picture 3.1. Our explanations for this approach are based on [DMSoo].

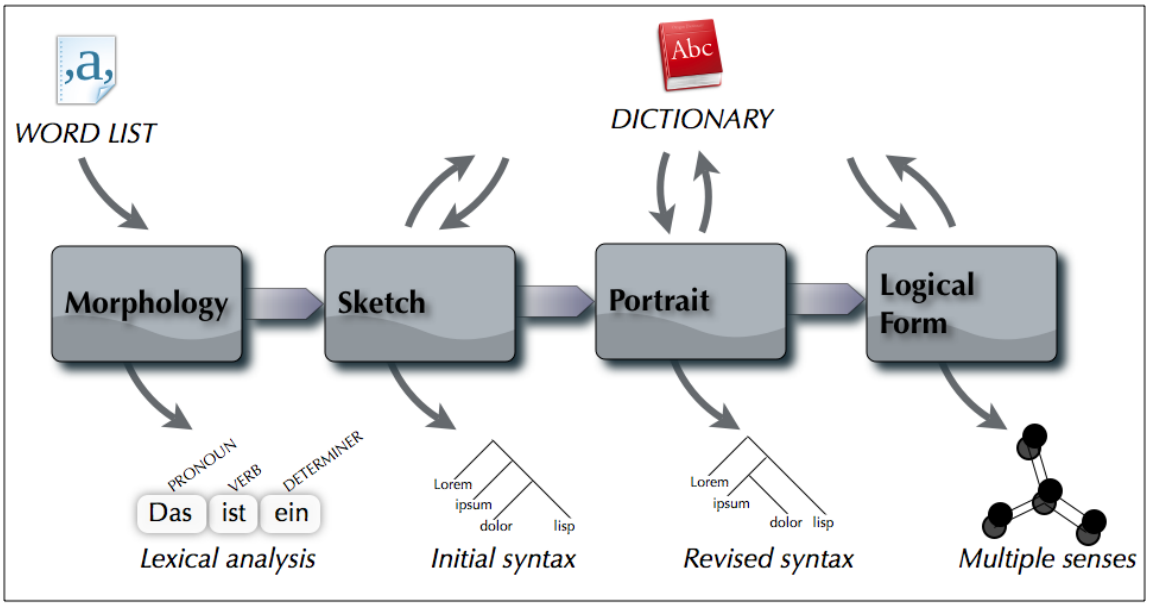

Figure 3.1.: Microsoft NLP system

1. The first stage is a lexical analysis of the text. The text is broken down into tokens which are mainly words and punctuation. Every token is morphosyntactically analyzed and a dictionary lookup is done. Multiword prepositions—e.g. in front of—are handled separately. Instead of storing every single word of those multiwords, they are stored as one entry in the dictionary. In this stage, two more types of tokens are considered: factoids and captoids. Factoids usually consist of dates and places whereas captoids consist of words in capital letters. When the processing is finished the output of this stage is a list of part-of-speech records.
2. The second stage is the parsing. In [DMSoo] it is also called syntactic sketch, see Figure 3.1. Augmented phrase structure grammar (APSG) rules are applied to build up a derivation tree. The output of this stage are one or more trees. Each node is a segment record of attribute-value pairs which describes the text that it covers.
3. In the third stage a refinement of the tree from stage 2 is done. This produces more reasonable attachments for modifiers like relative clauses. It is possible to move a prepositional phrase of time which is obvious to clause level by just using syntactic

information. A semantic reattachment is done in the Microsoft NLP system but it is not used in the Word 97 grammar checker yet.

4. The fourth stage is used to produce logical forms. The logical forms make explicit the underlying semantics of the text. Some phenomena is treated here, e.g. extraposition, long-distance attachment, and intrasentential anaphora. This stage is used under certain conditions, which is the case if the sentence is in passive voice or if it contains a relative pronoun.

3.1.2. LanguageTool for Openffice

LanguageTool [Lanb] can be integrated into OpenOffice [Ope]. Here, the processing is less complex than the approach from Microsoft. The development team splits up the text into sentences [Lana]. Like the grammar checker in Microsoft Word 97, all words in the sentences are separated and annotated with part-of-speech. Finally the text is matched against rules which are already built into the system. Furthermore rules given in XML-files are also checked.

3.2. Statistical Approaches

Some tried different approaches for grammar checkers based on statistical data. None of them are used as a productive tool. In the following we describe two approaches which have some aspects in common with our approach in LISGrammarChecker.

3.2.1. Differential Grammar Checker

In A Statistical Grammar Checker by Kernick and Powers [KP96] several methods for grammar checking are reviewed. In their opinion, most approaches start from the wrong site by looking at a sentence and checking its correctness. Kernick and Powers propose an approach which regards two sentences and tries to classify which one is more likely to be correct.

Their approach is based on statistics without a pre-existing grammar. They gain all statistical information from a corpus with more than 100 million words. This corpus is built from articles like Ziff-Davis, the Wall Street Journal and AP Newswire. They mainly use non-fiction texts and text that approximates semi-formal spoken English.

A first thought was to use semantics to process statistical information. But, as this was proved unsuccessful, they decided to review syntax instead. They tried to look for a minimal set of contexts, which can be used to distinguish words. An easy example of such a context

is form and from. Their final result is a differential grammar, which means *"a set of contexts that differentiate statistically between target words"* [KP96].

3.2.2. n-gram based approach

The approach called N-gram based Statistical Grammar Checker for Bangla and English is introduced in the paper [AUK06] written by Alam, UzZaman and Khan. Their system to check grammar using statistics is based on the assumption that the correctness of a sentence can be derived just from probabilities from all trigrams in a sentence. They want to use trigrams of words, but because of the used corpus, which is too small, they use trigrams of tags. To clarify their approach, we show a little example using bigrams. We have taken this example from their paper [AUK06], where bigrams are used, although the system is explained to work on trigrams. The example uses the sentence "He is playing.".

The first step is the calculation of the probabilities of all bigrams of the example sentence. The probabilities of all bigrams are multiplied. Their result is the probability of the correctness of the whole sentence.

Correctness Probability for the Example Sentence

P (He is playing.) = P (He | <start>) * P (is | He) * P (playing | is) * P (. | playing)

Deducting from the paper we assume that the probabilities of all pairs are gained by performing an n-gram analysis. For our example, this means going through corpora using a window of two, and determine how often each pair is found. The probabilities are therefore calculated by dividing the amount of the bigram through the amount of all bigrams.

As the last step, the probability P is tested if it is higher than some threshold. If this is true, the sentence is considered to be correct. Because of the multiplication of all probabilities only one missing bigram causes a probability of zero for the whole sentence. As described in the paper, this can occur often because of a too small training corpus which was used beforehand to train the system and, therefore, does not contain all word bigrams of the target language.

The concrete approach differs from the example. Instead of using the words itself, the part-of-speech tags corresponding to the words are used. This procedure is used because a lack of training data. The authors assume that they can get all trigrams if they are using part-of-speech tags. In addition, trigrams are used instead of bigrams. The threshold is set to zero. This means that every sentence which yields a higher probability than zero is considered to be correct.

The paper does not describe in detail which corpus was used to train the system. For the tagging, the part-of-speech tagger from Brill [Bri94] is used. There is no further information about the accuracy.

The paper does not clearly state how the accuracy measurements are achieved. For English—using a manual tagging method—the performance is denoted as 63%. This performance is assessed from the amount of sentences which are detected as correct out of 866 sentences which the authors denoted to be correct. There is no measurement for false positives. Furthermore the paper is imprecise which sentences were used to determine performance.

3.3. Our Approach: LISGrammarChecker

Our approach embodies a Language Independent Statistical Grammar Checker—we call it LISGrammarChecker. The main idea of our approach is based on statistics. We consider a database containing all bi-, tri-, quad-, and pentagrams of tokens and of tags of a language. This database is built up in a training phase. When the database is built up, we gather statistical information. We want to use this statistical information, i.e. n-grams, for finding errors and propose error corrections. Every sentence is checked for all n-grams of tokens and n-grams of tags and different error points are given if a specific n-gram is wrong.

Let us explain our approach with an example using trigrams only. For the sentence "All modern houses are usually very secure.", our approach will extract every trigram out of the sample sentence, e.g. "All modern houses", "modern houses are", "houses are usually", etc. These trigrams are saved into the database for training. While checking the sentence "All modern houses is usually very secure." (wrong verb tense), we check the existence of every trigram in the database. In this case, the trigram "houses is usually" is not found in the database, and thus an error is assumed. Additionally, this trigram analysis is also done for trigrams of tags, not only for the trigrams of the tokens themselves. A more detailed description follows in the next chapters.

Compared to the rule-based grammar checkers, our approach aims to be completely language independent. We achieve the language independence by the opportunity to train the database for every natural language. The checking is then done for the specified language.

Furthermore, we do not use any rules at all. We only use statistical data for grammar checking. This facilitates all languages to be used in the same program. And this program allows also languages where it is impossible or where the language is not spread enough to write a set of rules for grammar checking.

We do not use a differential grammar. Thus our approach differs from the described grammar checker from Kernick and Powers [KP96].

Compared to the described approach from Alam, UzZaman and Khan [AUK06] which also uses n-grams, our system uses more n-grams and combines them in a certain manner. The other approach only uses trigrams of tags to check the grammar. Furthermore, they take probabilities into account and there is no possibility to mark an error. If one trigram is not found, the whole sentence is just marked as incorrect due to the multiplication of the single probabilities of all trigrams. Instead we use bi- up to pentagrams and both n-grams of tokens and n-grams of tags. We also consider all tags of a sentence. We think that regarding only trigrams is not sufficient enough, because the word spheres are usually bigger than three and these are necessary to foresee all dependencies of a specific word. If we look at the wrong sentence "He and the girl is curious.", we would consider the whole sentence as correct if we are using trigrams only. The reason is that the verb is is not compared with the first noun he and thus the plural cannot be detected to be necessary. When using pentagrams, this is detected with the pentagram "He and the girl is". This pentagram would be considered as wrong. We do not use the probabilities of the n-grams directly but use an error counter with different weights for individual errors.

Part II.

Statistical Grammar Checking

4 Requirements Analysis

This chapter describes our approach. We start by explaining our idea and concept, and continue to develop the requirements for LISGrammarChecker. According to these requirements, we analyze the consequences for our work and start to specify what we need to consider. We analyze what would fit best to fulfill our requirements. We indicate which programming language we use, which corpora we prefer and how we obtain our statistical data. We regard the aspects of tagging within our grammar checker. Furthermore, we analyze how to process and store data in an appropriate manner.

4.1. Basic Concept and Idea

LISGrammarChecker uses statistical data to detect errors in a text. Therefore, we need a lot of texts which are used to build up the statistical database. These texts are processed in a certain way and the result is stored permanently. These stored results in the database can be extended at any time with new texts. We have two different approaches implemented in parallel. The one which we explain first is based on n-grams and is used for the main grammar checking. The second uses morphosyntactic features, i.e. word classes, and tackles therefore some specific grammar problems.

4.1.1. n-gram Checking

Our main approach for statistical grammar checking uses n-grams. In later sections we will summarize all this functionality as n-gram checking.

Data gathering To perform n-gram checking we need two different types of data: n-grams of tokens and n-grams of tags (see section 2.1.5). Getting the tags for each token in the texts requires the use of a part-of-speech tagger. From the input texts, we extract n-grams of 2 up to 5 consecutive tokens—bigrams up to pentagrams—and separately the bi- to pentagrams of the corresponding tags. The first step of our concept is thus the processing of a large amount of text. The results of this training step are stored persistently for further processing.

Token n-gram check In the next step we want to perform the grammar checking using the stored statistical data. Therefore, we compare an input text, in which errors should be detected, with the statistical data. To allow the comparison, the input text also needs to be analyzed with respect to its n-grams. Figure 4.1 shows an example, beginning in step 1 with the wrong input sentence "All modern houses are usually vary secure.". Step 2 shows examples for extracted n-grams. These extracted n-grams are just a small part of the n-grams in the sentence. A complete token n-gram extraction would start with all pentagrams of tokens for the whole sentence. Then we continue with the corresponding quadgrams of tokens, going on with the trigrams of tokens, and so on.

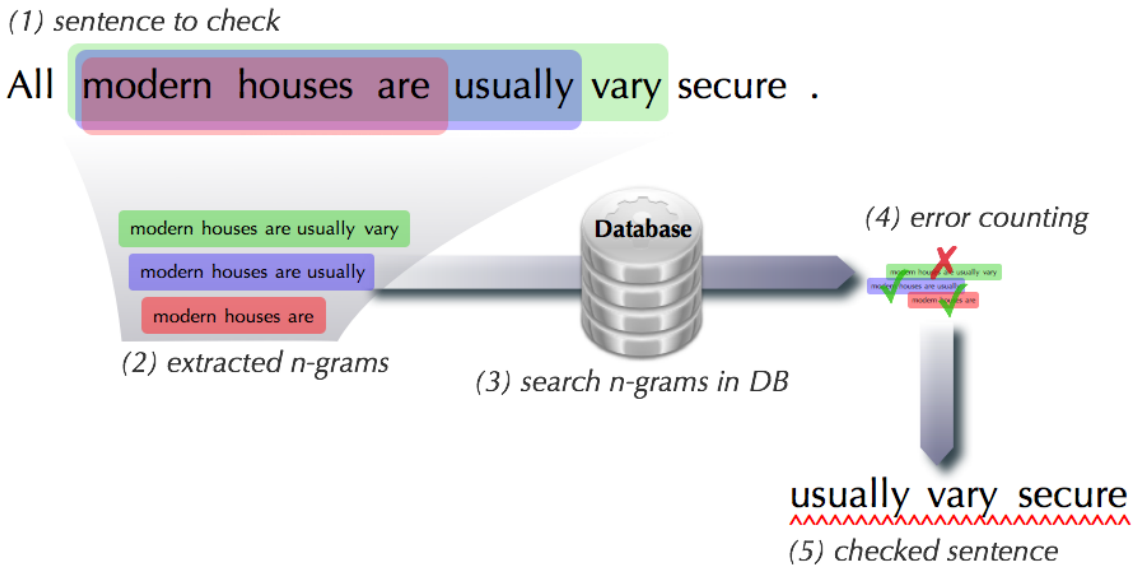

Figure 4.1.: Token n-gram check example

After that, all obtained n-grams are looked up in the database (step 3 in Figure 4.1). If an n-gram is not found in the database, it is assumed that this n-gram is wrong. In our example the pentagram "modern houses are usually vary" is not found in the database (see step 4). An error level is calculated corresponding to the amount of n-grams which are not found in the database. The smallest erroneous n-gram finally points to the error in the input text. In this example, this would be the n-gram "usually vary secure" and is therefore marked as an error (step 5).

Tag n-gram check Independent of the n-gram check of tokens, we analyze also the n-grams of the tags. This works similar to the described method with the n-grams of tokens, but—in addition to the bi- up to pentagrams—we use a whole sentence as one n-gram of tags. Thus we start with the analysis of the tags for a whole sentence. Then we continue with the pentagrams of the tags followed by the check of the quadgrams, trigrams, and bigrams as described above.

Furthermore, we consider three more n-grams—hybrid n-grams—which consist of tokens and tags. The first of them is a bigram consisting of a token and a tag on its right side. The second is almost the same but vice-versa: a bigram that consists of a token and a tag as its left neighbor. The third can be considered as a combination of the two other bigrams: it consists of a token in the middle and a tag on each side as its left and right neighbor.

Internet n-gram check An additional functionality, which can be activated along with the n-gram check of the tokens, is the use of n-grams from an Internet search engine, e.g. Google [Gooa] or Yahoo [Yah]. This option offers the possibility to gain n-grams from an indefinitely large amount of statistical data. If this functionality is activated, we extend the token n-gram check to use the data from Internet search engines to increase the amount of statistical data. That means, if there is an error in the token pentagrams from our database, the next step is a token pentagram check in the Internet, before we continue with the quadgram check from the local storage.

Error counting The error counting is done in step 4 of Figure 4.1. For both methods—n-gram of tokens and n-gram of tags check—there are individual weights for each error assumption. This means that the weights of how much an error assumption counts can be defined individually. For example, there is an individual weight for an error assumption of an unknown token pentagram, a weight for a missing token quadgram, a weight for a missing tag quadgram, etc. All error assumptions are counted corresponding to their weights. All these errors are summed up and the result is compared to an overall error threshold. If it is higher than the threshold the sentence is marked as wrong.

Correction proposal After the input text is checked and all errors marked, correction proposals corresponding to the found errors should be given. Our idea to correct errors

also uses n-grams. We want to make a proposal for the most probable phrase. Therefore, we use the borders of n-grams, leaving out the error itself.

For example, we take again the found error "usually vary secure" of our example, see step 1 in Figure 4.2. We take the pentagram corresponding to the error and replace the entry in the middle with a wildcard, i.e. in this case "are usually * secure .". In step 2 we search in our database for pentagrams with the first (are), second (usually), fourth (secure) and fifth (.) entry as specified. The middle element in the pentagram (i.e. the third one) is a wildcard (*). We search for possible correction proposals. There are two different possibilities how the proposal can be chosen from the results. Either we propose the result with the most occurrences or the most similar entry (compared to the skipped token) which has still an appropriate amount of occurrences. In step 3 of Figure 4.2, we propose "are usually very secure .". In this case, the proposal represents the phrase with the most occurrences, which is also the most similar entry compared to the error (vary and very differ only in one letter).

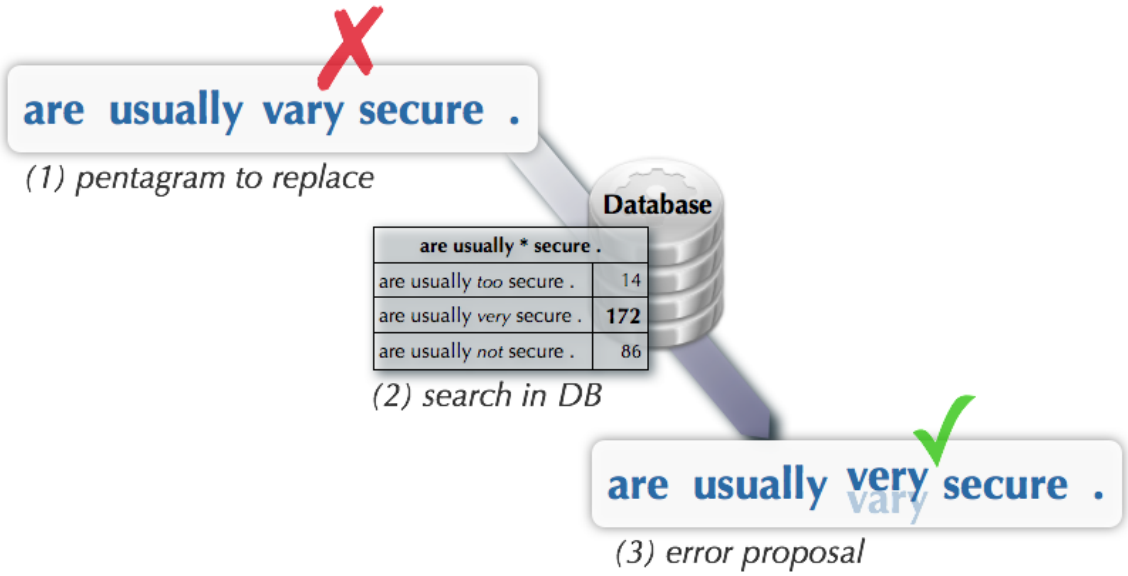

Figure 4.2.: Correction proposal example

4.1.2. Word Class Agreements

Our second approach does not represent a full grammar check but a solution to tackle two specific grammar problems. The idea is similar to the n-gram approach. Thus there is also

a lot of statistical data required. Again, we need a tagger to get the word classes, i.e. several tags that correspond to these word classes.

We check the agreement of a temporal adverb with the tense of the corresponding verb, and the agreement of an adjective to its corresponding noun. Therefore, we save this data while statistical data is processed and check a new text, where errors should be detected, against it.

Adverb-verb-agreement We save all temporal adverbs as proper tokens, e.g. yesterday, and the verb tags in a sentence. If there is the verb stayed in the sentence, we save its tag verb (past tense).

Adjective-noun-agreement Here we save all adjectives along with the noun which is described by the adjective. Both are used as tokens itself, e.g. we save the adjective young together with the noun girl.

From now on we refer to this second part of our approach as word class agreements. In this part we necessarily need a tagger which gives the word class tags. In addition, the tags that are used to detect the tokens need to be specified for every language, or, more precisely, even for every possible tagset. That means if we use the Penn Treebank tagset we need to specify that the tag RB marks adverbs. These little rules need to be defined because it is not possible to identify a certain word class just by looking at text. When checking sentences, the assumed errors are also summed up.

4.1.3. Language Independence

Our approach is language independent, because the statistical input texts can be given in any language. There could be a problem concerning language independence, if there exists no tagger for a specific language. This lack of a tagger could hinder the full functionality, i.e. the n-grams of tags and the second part of our idea using the word classes cannot be used. However, the n-grams of tokens from storage and from the Internet can be used in every natural language. A limitation to the language independency regarding the word class agreements could be the nonexistence of one or both features in a certain language but nevertheless they can be used in all languages where these agreement test make sense.

4.2. Requirements for Grammar Checking with Statistics

Our purpose is to build a language independent statistical grammar checker—LISGrammarChecker—which detects as many errors as possible and is capable of proposing an appropriate correction. Nowadays, all well-known grammar checkers use sets of rules to perform

these tasks. A big disadvantage in using rules is the lack of language independence. As specified in our definition of grammar (section 2.1.1), it is rarely possible to use the same rule for more than one language. Furthermore, some languages are too complex to be mapped into a set of rules, or the language is not spoken by enough people so that it is not profitable enough to build up a rule set for it. We try to counteract these problems by using statistical data to perform the grammar checking instead of rules. The data for every language needs to be processed separately because of their differences.

The use of statistical data instead of rules introduces some new challenges. Rules can describe many grammatical features and thus cover complex grammatical structures. This means that all rules for a complete grammar usually do not need so much storage memory. For example, the Microsoft Word 97 grammar checker uses about 3 MiB storage capacity [DMS00]. This is contrary to the statistical approach. Here we need the possibility to process and store a lot of data. Statistical data get more accurate and reliable, if more data are available at once. To get more data, the Internet n-gram feature can be activated. This requires LISGrammarChecker to query the Internet and handle the query results.

All the data needs to be stored in a usable and logical form into a database. The access time to the data must be fast enough to fulfill some speed requirements. It has to be fast enough to check the grammar in acceptable time. There are two competing factors—speed versus accuracy. Due to the problem of data handling, the speed of data retrieval decreases rapidly if the amount of data gets too large.

A very important constraint for this approach demands the statistical data to be (grammatically) correct. Only if this is fulfilled, will a sufficient statistical significance be reached and the information be considered reliable. It is therefore not acceptable to process and save data from every input sentence while grammar checking is done. Thus, we need separate modes in the program—one for data gathering, i.e. a training mode, and another mode using this database to perform the grammar checking on new input text which is possibly wrong, i.e. a grammar checking mode.

Our approach uses tags. We have the choice between developing our own tagger or using an existing one to get the tags for each token automatically. As we considered using decent taggers which are already language independent, we want those to be executed within the program. Like the statistical data itself, the tagging result is required to be correct because text which is tagged wrong can not be found in the data storage and logically the overall result is incorrect. The goal of full tagging accuracy is not reachable. As a single tagger is always less accurate than a combined tagger, taggers with good accuracies have to be combined to maximize the tagging accuracy.

The user needs to communicate with the system to provide the input text and to change preferences (mode etc.). Thus the user interface must provide possibilities to fetch the required informations. In addition, there is the demand for the grammar checker to show results.

The user need to see what the grammar checker has done so far, e.g. in training mode if the training is successful, or in checking mode if the input text contains errors and where they occurred (grammar checking result).

To use statistical data from training input for later grammar checking, the data need to be extracted and prepared using some logic. It must be specified what needs to be extracted and an algorithm is needed to do that work.

To perform grammar checking with statistical data, a logic is needed. The grammar checking is implemented by an algorithm. This logic needs to consider e.g. how the stored statistical data can be used to check the grammar of input text.

There is always a trade-off between the amount of detected errors in a text and the misleadingly marked errors, the so called false positives. The goal is to detect as many errors as possible. If there are too many false positives, the usefulness decreases. Thus the false positive rate should be as small as possible. We want to achieve this goal with a concept of thresholds, i.e. we do not intend to mark an error immediately but collect a reasonable amount of error assumptions to state the existence of an error. Along with the detection of an error comes the correction proposal. We want to gain these proposals from the most likely alternatives out of the statistical data.

Table 4.1 summarizes the developed requirements with their consequences for the implementation of LISGrammarChecker. In the following sections we analyze how to satisfy these requirements.

4.3. Programming Language

One main decision to implement an approach is the selection of the programming language. Demands for our implementation extend from the possibility to combine the execution of other programs as taggers to the use of a data storage system to save statistical data. It should have the opportunity to query Internet resources, e.g. to build a connection to an Internet search engine to get search results from it. For our approach it is necessary to operate with strings in many ways, which makes simple string handling valuable. The speed of program execution does not seem as important at a first glance. But if we consider the amount of data which is processed, the trade-off between speed and accuracy quickly raises in importance. This prefers a language which can be compiled to native machine code and can thus be executed directly without a virtual machine. Scripting languages are also not sufficient with regard to the execution speed. The paradigm of the language and features like garbage collection does not really matter. Demands like operating system and platform independence are not important for us.

Table 4.1.: The emerged requirements with their consequences

Requirements	Consequences
Language independence	Process statistical data separately for every language
	Save huge data permanently (database system)
Gain data and save it permanently	Separate training mode to save data
Grammar checking without saving wrong input	Separate grammar checking mode
Correct statistical data	Gain correct text from reliable sources
Program execution speed	Fast program components (programming language)
	Short data access time (fast data storage)
Accurate reliable data	Much statistical data
Use Internet n-grams	Possibility for Internet queries
Tagged input text with high accuracy	Integrated tagger
	Combined tagger
User need to interact with the system	Appropriate user interface to give input and preferences
Show results (training successful or errors found)	Output result to the user
Save data in training mode for later grammar checking	Algorithm to extract information from input
Perform grammar checking in grammar checking mode	Algorithm to perform grammar checking
Few false positives	Use thresholds for error calculation
Propose a correction	Gain proposals from the most likely alternatives out of the statistical data

There are various languages which meet the needs to different extents. As we consider the demands of a fast program execution, the possibility to run external programs, and simple string handling as important, we decided to use the rather new programming language D [Dig]. Even though it is a very young programming language, it is very well supported through a front-end for the GNU Compiler Collection called GDC (GNU D Compiler) [Fri].

D does not need a virtual machine like Java, nor is it a scripting language as PHP or Python. Native binaries raise the speed of program execution compared to these languages. The programming language is not bound to object-orientation. Functional programming which can raise the speed of program execution is also supported. String handling is supported in a sophisticated way like e.g. in Java, which is an advantage over C or C++. It is possible to execute other programs within D and handle their standard input (stdin) and standard output (stdout). D is similar to C and C++, thus it is easy to adopt. Additionally, C code can directly be included and mixed within D code. In case of missing D functions or bindings (i.e. MySQL [MS]), the C alternatives can be used. D comes with a garbage collector which is a nice but not necessary feature for our approach.

4.4. Data Processing with POSIX-Shells

A lot of data processing is needed before the proper grammar checking can be done. This means that data has to be processed and converted to the format which is expected by the program. To ensure the correct delivery of data to the program we use the capabilities of POSIX-shells [IEE04].

An important point is the use of the shell-based streams standard in- and output. On the one hand, LISGrammarChecker itself needs to be used with a shell command. It needs to handle huge input texts, thus the standard input is both suitable and useful, because it is fast and accurate. Furthermore, to offer the possibility to execute LISGrammarChecker within another program through piping (redirecting of streams) both the input and output of our grammar checker is handled through these standard streams (stdin and stdout). All capabilities of POSIX-tools, such as grep, sed, or echo, can be used in combination with LISGrammarChecker. On the other hand, the execution of other programs, especially PoS taggers, within LISGrammarChecker need to be provided through a command line call. The taggers can be included through the use of POSIX commands, in this case with the help of shell scripts. This works, because D supports the use of commands to execute other programs and handle their standard in- and output.

One important feature is the capability of scripting. In POSIX-shells there are a lot of programming language features available. Among them are if clauses and while and for loops. Used with the multifaceted POSIX-tools every data conversion can be done.

4.5. Tokenization

Our approach needs a good tokenization method to avoid unnecessary mistakes. The main issue is the sentence boundary detection. For our approach we propose to use a tokenizer

based on the theory by Schmid in his paper [Sch00]. This work aims to be mainly a period disambiguation method which achieves an accuracy higher than 99.5%. Schmid uses different features which denote the sentence boundaries. Among them are a dictionary of abbreviations and pattern.

4.6. Part-of-Speech Tagging

Part-of-speech taggers are very important for our approach. The process of tagging corpora is a presupposition for some functionality where tagged text is required. It is important to choose taggers which yield a high accuracy rate. Therefore, the taggers need to get pretrained files for all used languages. Furthermore, all taggers need to be fast to ensure that grammar checking is possible in sufficient time.

In LISGrammarChecker we include TnT [Bra00], TreeTagger [Sch94], and StanfordTagger [TM00] for English and use the Penn Treebank [MSM93] tagset. For German, we also use TnT, TreeTagger, and StanfordTagger, but instead of the Penn Treebank tagset we use the STTS tagset [STST99]. One requirement of LISGrammarChecker is language independence, thus it is easily possible to include more taggers for other languages. The TnT tagger for example yields an accuracy of 96,7% both in English and German with the provided models. The models were trained with newspaper texts. TreeTagger yields an accuracy of about 96.3% for English. We choose these taggers for our first prototype implementation, because they fulfill our requirements.

When comparing the tagsets, there is a contradiction if a tagset with less or more tags is chosen. The more tags a tagset consists of the more complex is the tagging and thus the accuracy lower, but the exactness of the classification is higher. As we have described above, we decided to use the smaller tagsets because they yield a higher accuracy.

4.6.1. Combination of PoS Taggers

Accuracy is one requirement for tagging. Combined tagging can produce higher accuracies than the use of only a single tagger. For combined tagging it is the best to use as many different tagger types as possible. Let's consider the example sentence "This shows a house." and do simple voting with three taggers A, B, and C. Table 4.2 shows the result.

If we look at tagger B we see that the word shows is accidently tagged wrong. The word shows can be a plural noun, which has the tag NNP. In this context this is wrong, because it is a verb in third person singular, i.e. the tag VBZ is correct. There would be a mistake in tagging if only tagger B is used, but with the simple voting combination the final result denotes the

Table 4.2.: Simple voting example

Tokens	Tagger A	Tagger B	Tagger C	Combination
This	DT	DT	DT	DT
shows	VBZ	NNP	VBZ	VBZ
a	DT	DT	DT	DT
house	NN	NN	NN	NN
.	SENT	SENT	SENT	SENT

word shows as VB because two of three taggers (the majority) propose that tag. All taggers make different errors and thus the combination can eliminate some tagging errors.

Regarding the trade-off between tagging accuracy and execution speed, we have to find the best trade-off for our intention. If it takes for example one millisecond to achieve 96% accuracy instead of 96.2% in five seconds, the 0.02% greater accuracy is not as valuable as more speed. For our prototype, the above mentioned taggers are sufficient. The simpler the combining method, the faster is its execution. Thus we start with the implementation of simple voting.

4.6.2. Issues with PoS Tagging

While using PoS tagging, there are several aspects which need to be considered. For example the encoding must be consistent. LISGrammarChecker uses UTF-8 [Yer03], thus the taggers need to support this.

In case of a combined tagging, the tokenization must be done the same way for all taggers. This is required to get correct results. One solution is that we do the tokenization for all taggers once and thus in the same way before the taggers get the texts.

The next problem which arises by using taggers is the input to the tagger and the output after tagging. This need to be considered and implemented in an adaptive way so that it is possible for LISGrammarChecker to support various kinds of taggers.

4.7. Statistical Data Sources

In order to get a database which allows an accurate grammar check, we need good statistical data for training. In this case good means that the used corpora and texts need to be of good quality. Good quality means grammatical correctness and good language style. It is therefore not possible to use every available corpus resource. Transcripts of recorded spoken text are not recommended, because it does not fulfill the quality demands. Useful for the English language are for example newspaper texts from ABC, NYT, or Bloomberg, and texts from the American National Corpus [IS06] (over 20 million words) as well as Wortschatz Universität Leipzig [QH06]. For German, Wortschatz Universität Leipzig is also useful. It contains 3 million sentences with about 15.73 words per sentence which are 47,190,300 words.

4.8. Data Storage

LISGrammarChecker needs to handle and store huge amounts of data. It must be able to handle more than 100 million different data sets. The requirement for a database is to store text phrases, e.g. n-grams, and the data needs to be ready for retrieving in a short period of time. To achieve these tasks efficiently, the data must be stored in an ordered way. Several methods can be used.

We reviewed three methods: a flat file, a XML file, and a database system. With regard to LISGrammarChecker, all three have advantages and disadvantages in their usefulness of necessary features. Some key features are compared in Table 4.3. The markers represent a scale of how well a specific feature is supported. The scale from best to poorest is: ++ (very good support), + (good support), 0 (neutral), - (poor support), and - - (very poor support).

If we regard the complete table, a database system would be the best solution for our prototype. We have a lot of scattered data which needs to be sorted in some way (e.g. there should be no double entries). Sequenced data means that the data is already structured and can be written like a stream. That would only be the case after our program has finished and all the data is processed. If we look at Table 4.3, we see that a database has only poor support for sequenced data, but a flat file supports sequenced data very well. This indicates that if the data has been structured once that the data can be copied like a stream to another database which is a lot faster than doing the extraction again. A database has advantages compared to a simple text file, especially optimization, sorting, and searching. We do not need to do this on our own in the program. The XML file has also some advantages which could be of use for us but the trade-off shows that we would rather use a database system for LISGrammarChecker. Many features from Table 4.3 are supported.

Table 4.3.: Comparison of several storing methods

Feature	Flat file	XML file	Database
Write sequenced data	++	+	- -
Write single set (part)	-	o	+
Organization of information	-	+	+
Access particular data	- -	-	++
Check if set exists before writing	- -	-	++
Optimize data structures	-	+	++
Sort and search data sets	-	+	++

The next question is which database to use. Again, there are several possibilities. We decide to use a MySQL [MS] database. It has the big advantage that there are MySQL bindings which allow a simple integration into D. Furthermore, MySQL is a well-known and reliable open source database system.

One advantage of using the database system MySQL is its capability to use clusters. That means it would be possible to use more than one machine to store the data on. This would improve the speed considerably. We will consider this for further improvements to the speed.

5 Design

In this chapter we give an overview of the design of LISGrammarChecker. We introduce all program components and show how they interact with each other.

5.1. Interaction of the Components

LISGrammarChecker reflects the structure of a typical computer program—it consists of an input, a processing, an output, and a database component. Figure 5.1 shows an overview of the general interaction of these abstract components. LISGrammarChecker is designed to process data in a serialized form, starting with an input, continuing with the processing part, and concluding with the output.

There are two different workflows possible: one to train the program with statistical data, training mode, and one for doing the checking of grammar, checking mode. In training mode, the input is statistical text, which is analyzed in the processing part and written to the database (see section 5.3 for more details). In checking mode, the input is also text. This

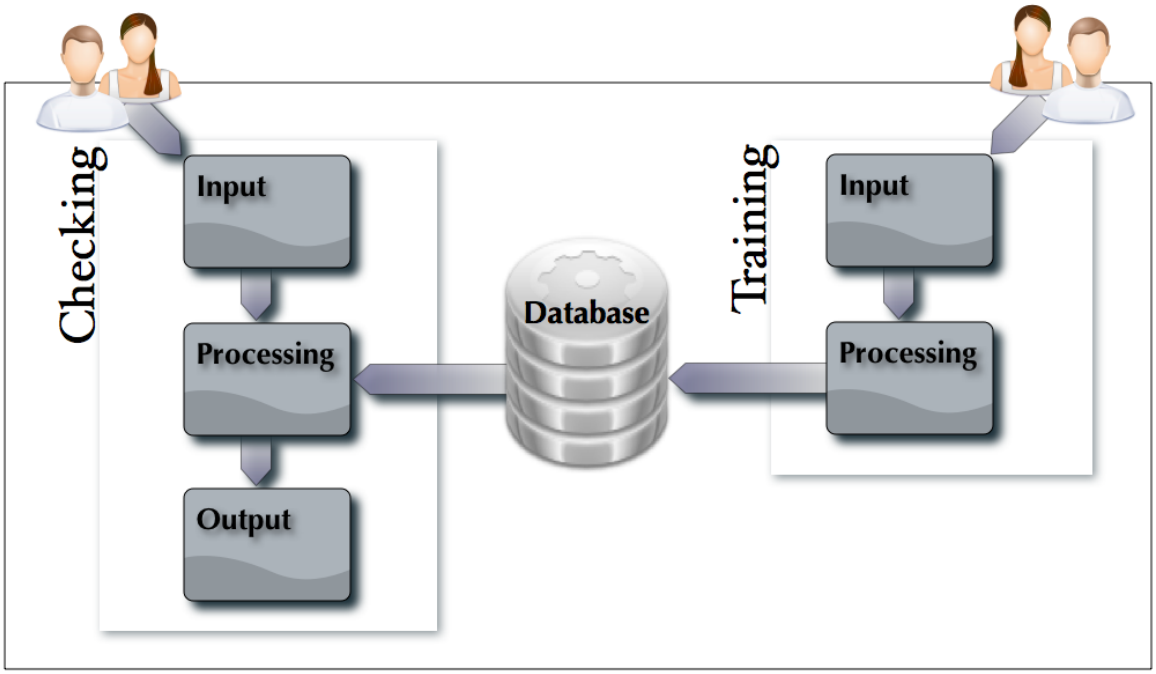

Figure 5.1.: Abstract workflow of LISGrammarChecker

text will be checked for errors in the processing part. The results from this grammar checking, e.g. the indication of errors, is given to the user as output. Section 5.4 describes the checking mode.

Before we give the description of these two modes, we consider the user interface with the in- and output of LISGrammarChecker.

5.2. User Interface: Input and Output

LISGrammarChecker is a shell program without a graphical user interface. Input and output of the program happens on the command line. This means that options and data input are given to the program by using the standard input (stdin).

Both training and checking mode have different inputs. In both modes, input parameters can be specified, e.g. the language, the tagging method (see section 5.5), or a database name (see section 5.6). In training mode, statistical data in form of (huge) texts are given as input to be stored in the database. Subsection 5.3.1 presents more details about the input in training

mode. In checking mode, a sentence or even a text is given as input, which is checked for grammatical errors. This is explained in subsection 5.4.1.

The output is given out on the standard output (stdout). It differs in both modes. The training mode outputs just a success message, that the extracted data has been written to the database. We do not indicate this output in our overview (Figure 5.1), because it is of no relevance for the main program workflow. The output from checking mode is much more important. It presents the grammar checking results, especially the error indication together with corresponding correction proposals. The results from the grammar checking part are explained in subsection 5.4.5.

5.3. Training Mode

The training mode is used to fill the database with statistical data. The training is a precondition before any text can be checked.

The first step of this mode is the input. Here the program expects texts, which are used to extend the database. This data is passed to the tagging component first, which is explained in section 5.5. After that, the tagged text is given into the data gathering component, where the needed phrases and information is extracted and afterwards written into the database for further processing. Figure 5.2 demonstrates the described training workflow. We continue explaining the input in the following subsection.

5.3.1. Input in Training Mode

All input is passed to the program by input parameters on the standard input. There are several input parameters available in training mode:

Statistical data This is the main and most important input to the program. Huge collections of text serve as input to train LISGrammarChecker, i.e. to extend the database with statistical data. It is generally possible to input all kind of texts, but they should fulfill some standards. It is, for example, very important that the texts are (grammatically) correct. Only if this is assured and enough texts are used to learn, will a sufficient statistical significance be reached and the information be reliable. Thus, it makes a difference which kind of input texts are used to fill the database.

Language The language can be specified by a parameter. This is used to assign the input data to the correct language. If the optional language parameter is missing, English is chosen as the default language. The language information is needed in order to store the input data in the correct database tables.

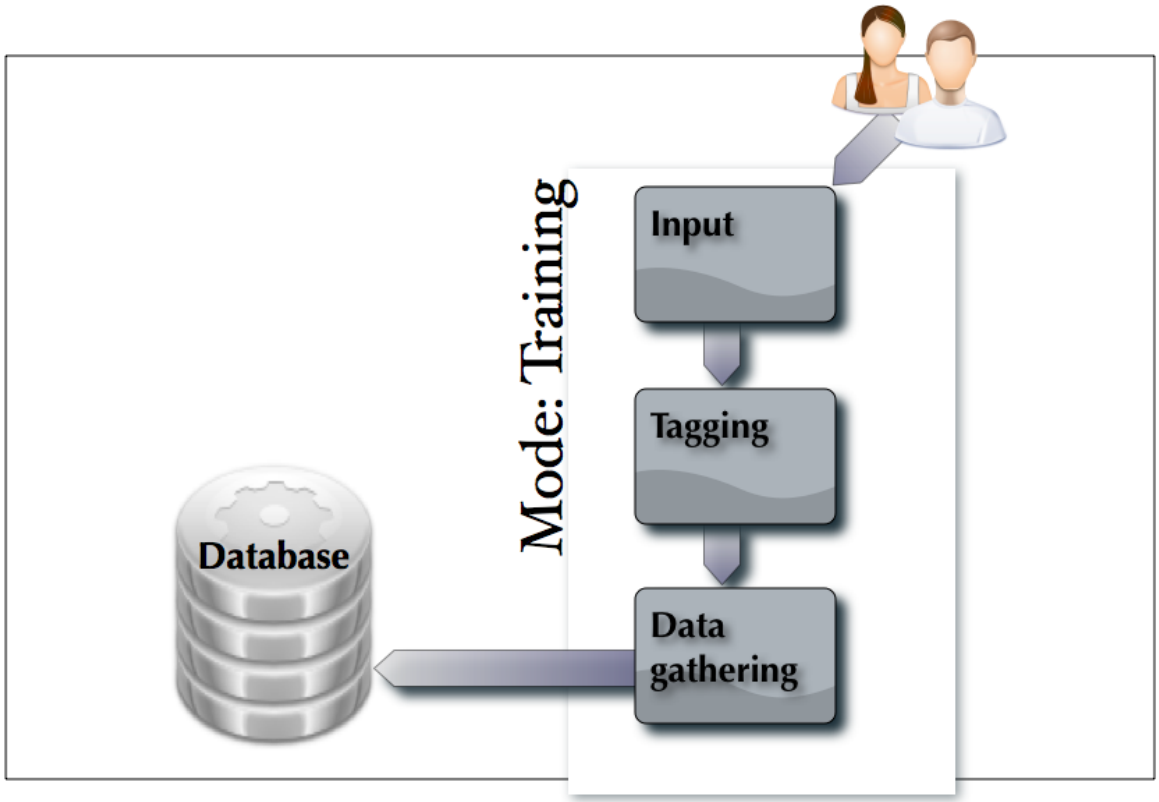

Figure 5.2.: Workflow in training mode

Tagging method Furthermore, the tagging behavior can be changed. The default uses all available taggers for the chosen language and their outputs are combined by a simple voting algorithm. By setting this parameter, the combined tagging can be skipped in favor of one single tagger. See section 5.5 for more details about the possible tagging methods.

Database name An individual database name can be specified. This is optional and usually a default name is used. This option offers the possibility to build up a new database without harming another. Using this option, the databases can be used for different advantages.

5.3.2. Data Gathering

The data gathering step analyzes the input texts and extracts statistical information from them. These data are stored in the database. Later, they are used for the grammar checking.

In this step the texts are already tagged.

We analyze the input texts in two different ways: regarding their n-grams (first part of our approach) and regarding their word class agreements (second part). While parsing the texts, we extract all needed information at once. For every entry that is stored to the database, a corresponding amount of occurrences are saved. This amount describes how often a specific data set occurred. See section 5.6 for more details about the form of the stored data.

n-gram checking The first and main part of our approach uses n-grams as explained in section 4.1.1. We use both the n-grams of tokens and n-grams of tags. Token n-grams means that we take a look at n-grams of two up to five neighbored tokens (all bigrams to pentagrams). While performing the n-gram analysis we extract all these token n-grams and store them in our database. We do the same with the tag n-grams. The only difference to the treatment of the token n-grams is the usage of tags instead of the tokens themselves. As a speciality we also store the tag structure of a whole sentence; this means that we save all tags of each sentence as a tag combination.

Figure 5.3 illustrates an example on the basis of the sample sentence "All modern houses are usually very secure."—the two trigrams "modern houses are" and "houses are usually" are taken from the sentence and stored to the database. This example sentence has more than two trigrams. If we take the formula from section 2.1.5 to calculate the amount of n-grams, we get six trigrams: k-(n-1) = 8-(3-1) = 6, where k = 8 (tokens in the sentence) and n = 3 (classifies the n-gram as a trigram). The same formula can be taken to calculate the amounts of pentagrams, quadgrams and bigrams.

We also need to evaluate the token n-grams from the Internet and store them but we do not request any Internet n-grams in training mode. The reason for that is that it is not useful because the training data are considered as correct and needs therefore not to be double checked. But nevertheless we use them in checking mode, and every n-gram loaded from the Internet is stored locally to our database when it is requested.

Furthermore, we store three more n-grams of hybrid type. They represent a mixture of tokens and tags. The first is a bigram consisting of a token and a tag as its right neighbor. The second is almost the same but vice-versa: a bigram that consists of a token and a tag as its left neighbor. The third is the combination of both bigrams: a token in the middle of two tags.

Word class agreements The information needed for the second part of our approach, which is already explained in section 4.1.2, refers to the word classes. We take this information from the tags. This part of our approach does not represent a whole grammar checking logic. It rather tackles two specific grammar problems. In LISGrammarChecker, there are two kinds of word class agreements implemented: the agreement between an adverb and a verb as well as the agreement of an adjective and a noun.

Figure 5.3.: Two sample trigrams (of tokens) are stored into database

Figure 5.4 shows an example of an adverb-verb-agreement. The sample sentence is "Yesterday, the man stayed at home.". At first, this sentence is tagged by a tagger. In the figure, the tags are illustrated as word classes "adverb, determiner, noun, verb (past tense), preposition, noun, punctuation". For the adverb-verb-agreement, we need the adverbs as proper tokens and the tags of the verbs. This means that we parse the training input in a next step. If we find the tag adverb in a sentence, we save the token of the adverb together with the tags of the verbs in the same sentence. In our example, the extracted information is highlighted in blue in Figure 5.4: the token "Yesterday" and the tag "verb (past tense)". These data are stored to the database.

Figure 5.4.: Extract adverb and verb

The adjective-noun-agreement works similar to the adverb-verb-agreement. An example sentence "Das ist ein grünes Haus." is shown in Figure 5.5. This sentence is given as training input to LISGrammarChecker. Again we start by tagging the text. The tags which are illustrated as word classes in this case are "pronoun, verb, determiner, adjective, noun, punctuation". For the adjective-noun-agreement we need the adjectives together with the nouns that are described by the adjectives. We need both as proper tokens. In our example sentence, this is the adjective grünes, which describes the noun Haus. They are written in blue in Figure 5.5. Both tokens are stored to the database.

Figure 5.5.: Extract adjective and noun

Concluding to this data gathering subsection, Table 5.1 summarizes all data that we extract.

Table 5.1.: All information that is extracted in training mode

Checking method	Extracted data (assigned to methods)
Token n-gram check (Internet functionality not yet relevant)	Token bigrams
	Token trigrams
	Token quadgrams
	Token pentagrams
Tag n-gram check	n-gram of all tags of a sentence
	Tag bigrams
	Tag trigrams
	Tag quadgrams
	Tag pentagrams
	Bigrams of a token with its left neighbors tag
	Bigrams of a tokens with their right neighbors tag
	Trigram of a tokens with its left and right neighbors tags
Adverb-verb-agreement	Adverb tokens with belonging verb tags
Adjective-noun-agreement	Adjectives and belonging nouns, both as tokens

5.4. Grammar Checking Mode

As the name of this mode already proposes, the grammar checking itself is done here. This workflow starts also with the input: text, which will be checked. In addition to that, there can be input parameters to specify the language, the tagging method, and further details about the checking method. The general workflow for the grammar checking is shown in Figure 5.6. The tagging mechanism follows the input part, similar to the one in training mode. The figure shows that the checking methods and the error counting constitute the core grammar checking. These components interact with the database and optionally with the Internet. After the identification of all errors, the correction proposal component comes into play. The identified errors together with their correction proposals are presented to the user in the output component. The following subsections explain the checking mode in more detail, starting with the input.

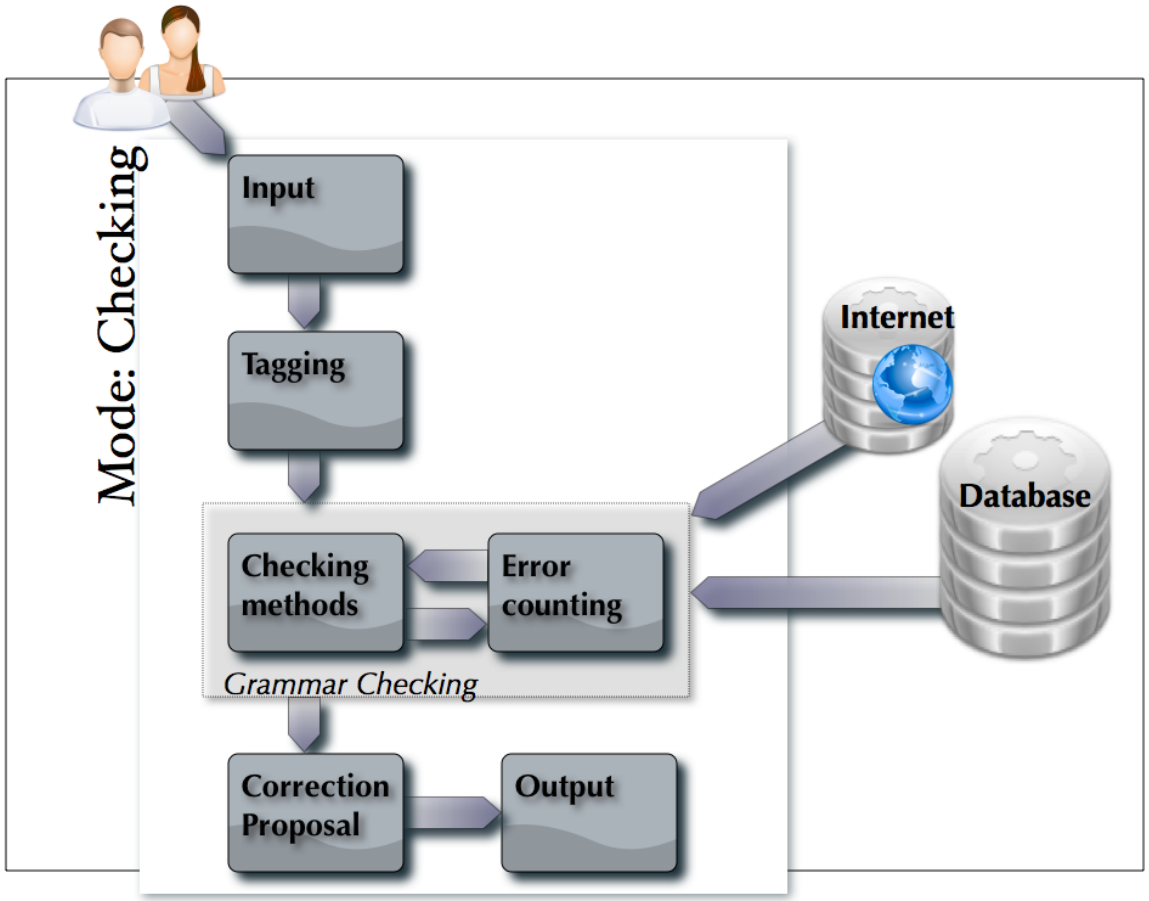

Figure 5.6.: Workflow in checking mode

5.4.1. Input in Checking Mode

There are several input parameters for the shell, which can be used in checking mode:

Input text The grammar checking mode requires text as input which should be checked. This texts can be a single sentence or a whole text. These texts are necessary for checking mode and must be specified.

Language There is an optional parameter which specifies the language. It is used to prepare parts of the program to use the correct language. If this language parameter is missing, English is chosen as default language. As in the training mode, the language information is needed in order to retrieve the correct data from the database.

Tagging method The tagging method can be specified using another input parameter. This parameter is optional. If it is not specified, all available taggers for the current language are used. Their results are combined using the simple voting algorithm.

Database name An individual database name can be set. This is optional and usually a default name is used. This option offers the possibility to use a different database with other statistical data.

Error threshold The error threshold can also be specified. This parameter defines the overall error threshold for all errors in one sentence. For more details about the error threshold and the error calculation see section 5.4.3. This parameter is set to 30 by default.

Internet n-grams The amount of available token n-grams can be extended by activating the Internet n-gram check. This is deactivated by default. This feature extends the token n-grams in the local database with n-grams that are searched in the Internet. The preferred search engine can be specified, e.g. Google [Gooa] or Yahoo [Yah]. Furthermore, a threshold can be given, which specifies the amount of search results that defines a positive search. This threshold is optional as well; its default is 400.

5.4.2. Grammar Checking Methods

The checking methods component, which is illustrated in Figure 5.7, gets tagged text as input. This text is now ready to be checked for errors. Various properties that can reveal errors are examined in several checking methods. All checking methods can detect distinct errors. At first, all detected errors are only handled as error assumptions. A true error is defined when enough error assumptions occur. True errors are calculated in the error counting component (see below). As the checking methods are already described in detail in section 4.1, we briefly recall them:

Token n-gram check Use the n-grams of token from database for checking. First, all pentagrams of token in the sentence are checked. In case of one or more pentagrams are not found, the token quadgrams come next, then the trigrams and finally the bigrams.

Internet functionality This is no complete checking method, but rather an extension for the token n-gram check which can be activated optionally. In this case, the statistical data is extended with n-gram search results from an Internet search engine. The search engine can be specified, e.g. Google [Gooa], Yahoo [Yah], or Google Scholar [Goob]. If this functionality is activated and an error in a pentagram of token occurred, the pentagram is sent to an Internet search machine. The search machine returns the amount of found results. If this amount is greater than a certain threshold we define this pentagram as correct. After that we continue with the quadgrams of token from our database. Here the same rule applies: if a quadgram is wrong, an Internet search is started for that particular quadgram. The same will be the case for tri- and bigrams. All requested results are stored in the local database, so that the same request does not need to be sent twice. This functionality is deactivated by default because of the numerous requests to the Internet.

Tag n-gram check Uses the tag n-grams from database for checking. It starts with the tags of a whole sentence, and continues with the pentagrams of tags. If a pentagram is not found we check the quadgrams etc. Furthermore the n-grams of the hybrid tags and tokens are checked.

Adverb-verb-agreement If an adverb is found in the sentence, the agreement between it and the verb tag is checked. To do that, the token of the adverb is used. For the verbs (we use them all because some tenses consist of more than one verb) we use the tags. A lookup for that combination is done to check if it is valid.

Adjective-noun-agreement If there is an adjective in front of a noun in the sentence, the agreement of the noun and the describing adjective are checked. Here, for both the adjective and the noun, the tokens are compared.

All checking methods are executed in one pass but all need different statistical informations. Each checking method component interacts with the database and, if required, with the Internet component.

Figure 5.7 shows the interaction of all components. First, the individual grammar checking methods are executed. While they are working, they determine missing items, e.g. not found n-grams. If an item is not found in the database we make an error assumption. These error assumptions are stored for further processing. We store the type of the missing item together with its position, so that we remember the exact location, where the error assumption occurred and of which type it is. These data are necessary to calculate the overall errors

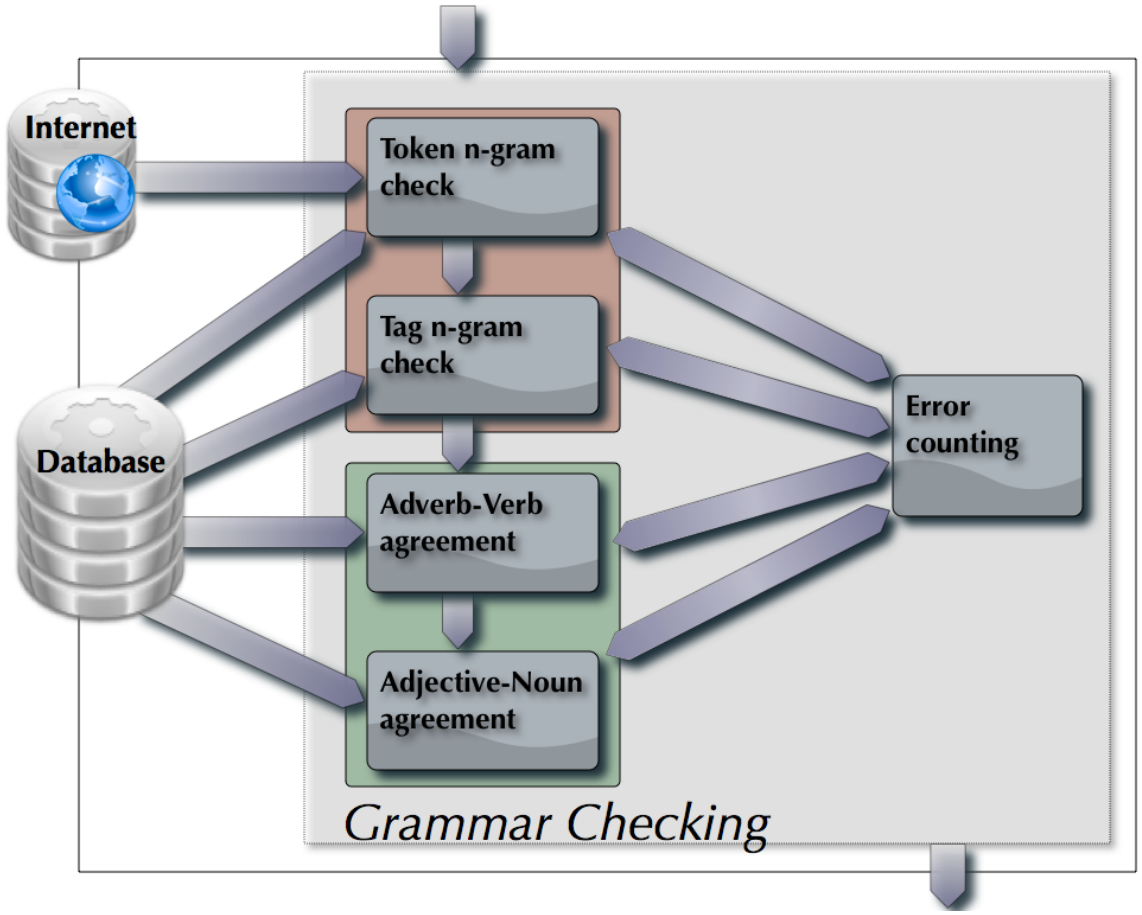

Figure 5.7.: Grammar checking

and to point out the phrase where the error occurred. When the true errors are identified, corresponding corrections proposals can be evaluated.

As all checking methods use different approaches to achieve their results, they all determine different error assumptions. These error assumptions can be weighted individually to calculate an overall error result.

5.4.3. Error Counting

All grammar checking methods which are explained in the previous subsection detect different errors. These errors are handled as error assumptions at the beginning. The checking methods store the detected errors assumptions with their type and location for further treatment. The error counting component uses this stored error assumption information

to calculate the overall error. Table 5.2 shows all possible error assumptions that can be determined by the checking methods. These error assumptions occur, if an entry is not found in the database. For example, if the pentagram of tokens "modern houses is usually very" is nonexistent in the database, an error is assumed.

Table 5.2.: All possible error assumption types

Checking method	Possibly determined error assumptions
Word n-gram check (Internet functionality deactivated)	Nonexistent token bigram
	Nonexistent token trigram
	Nonexistent token quadgram
	Nonexistent token pentagram
Word n-gram check, Internet extension activated	Nonexistent token bigram
	Nonexistent token trigram
	Nonexistent token quadgram
	Nonexistent token pentagram
	Nonexistent token bigram from Internet search
	Nonexistent token trigram from Internet search
	Nonexistent token quadgram from Internet search
	Nonexistent token pentagram from Internet search
Tag n-gram check	Nonexistent bigram of a token with its left neighbor tag
	Nonexistent bigram of a token with its right neighbor tag
	Nonexistent trigram of a token with its left and right neighbor tags
	Nonexistent n-gram of all tags of a sentence
	Nonexistent tag bigram
	Nonexistent tag trigram
	Nonexistent tag quadgram
	Nonexistent tag pentagram
Adverb-verb-agreement	Nonexistent adverb-verb-agreement
Adjective-noun-agreement	Nonexistent adjective-noun-agreement

All error assumptions can have individual weights. These weights can be specified within

the program. The overall error calculation sums up all error assumptions and the result is an overall error amount which denotes the correctness of a sentence. If this amount is above a specified error threshold, the sentence considered as wrong.

The calculated error is graded by the amount of found combinations. A hierarchy can be built up, where errors with larger n-grams weigh less than short n-grams, e.g. a nonexistent pentagram counts less than a nonexistent bigram. In addition, there can be a hierarchy for the relation of not found n-grams of tokens and n-grams of tags.

Even if there are few error assumptions in a sentence, the overall calculated error can be defined as "no error". This means that, if the overall calculated error is lower than the specified error threshold, the sentence is not defined as wrong. This approach aims to minimize the amount of false positives.

For a better understanding of the error calculation with individual weights, we give two examples. Both examples demonstrate token n-gram checks. The examples are consciously very similar, with the same error weights. They differ only in one trigram error assumption. Example one has a wrong trigram, and the overall result indicates an error. Example two has no wrong trigram, and this slight difference causes no overall error indicated in the second example.

Error Calculation Example One

Error assumptions 6 token pentagrams, 4 token quadgrams, and 1 token trigram.

Error weights token pentagram counts 1, token quadgram counts 5, and token trigram counts 10.

Error threshold 30

Overall error 6x1 (pentagrams) + 4x5 (quadgrams) + 1x10 (trigrams) = 36

Result As the overall error threshold (30) is lower than the overall calculated error (36), these error assumptions are indicated as a true error.

Error Calculation Example Two

Error assumptions 6 token pentagrams, and 4 token quadgrams.

Error weights token pentagram counts 1, and token quadgram counts 5.

Error threshold 30

Overall error 6x1 (pentagrams) + 4x5 (quadgrams) = 26

Result As the overall error threshold (30) is still higher than the overall calculated error (26), these error assumptions are not indicated as a true error.

5.4.4. Correction Proposal

After the identification of the true mistakes, a correction proposal is evaluated. We want to make a proposal for the most probable alternative phrase out of the statistical data. Therefore, we use the token n-grams again.

In detail, we use the token n-grams which include the mistake. The borders from these n-grams are those from the wrong n-grams. The token in the middle of the n-gram, where the error is left out, is replaced by the most probable alternative.

For example, we take again the (wrong) sentence "Modern houses are usually vary secure." with the detected error "usually vary secure", see step (1) in Figure 5.8. Then we use the surrounding pentagram with a wildcard for the error itself, i.e. in this case "are usually * secure .". In step (2) we search in our database for pentagrams with the first (are), second (usually), fourth (secure) and fifth (.) entry as specified. The middle element in the pentagram (i.e. the third one) is a wildcard (*). We search for possible correction proposals. There are two different ways; either we propose the result with the most occurrences or the most similar entry (compared to the skipped token) which has still an appropriate amount of occurrences. In step (3) of Figure 5.8, we propose "are usually very secure .". In this case, the proposal represents the phrase with the most occurrences, which is also the most similar entry compared to the error (vary and very differ only in one letter).

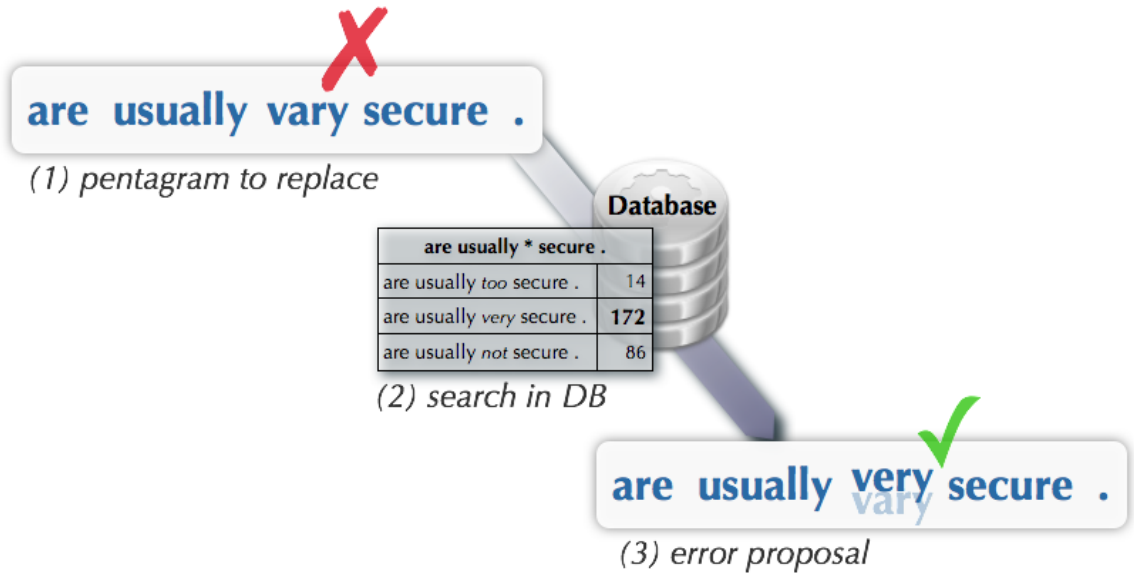

Figure 5.8.: Correction proposal example (repeated from Figure 4.2)

5.4.5. Grammar Checking Output

Grammar checking mode prints all grammar checking results to the standard output (stdout). This mainly includes:

Error assumptions All error assumptions that occur in any grammar checking method are shown. These error assumptions are very multifaceted. Table 5.2 lists all possible error assumptions that can occur.

True detected errors All error assumptions are counted with individual weights. If these overall calculated error proposals in one sentence are higher than a specified threshold, true errors are defined. LISGrammarChecker indicates these errors, i.e. points to the locations where the errors occur.

Correction proposals For all true mistakes that are marked, a correction proposal is given.

5.5. Tagging

Tagging is done in both modes—training and checking. It is always done in the early stages of the program workflow, prior to any other process. The texts which are given to LIS-

GrammarChecker serve as input to the tagging component. The output is tagged text. The tagging component is composed of smaller parts, as one can see in Figure 5.9.

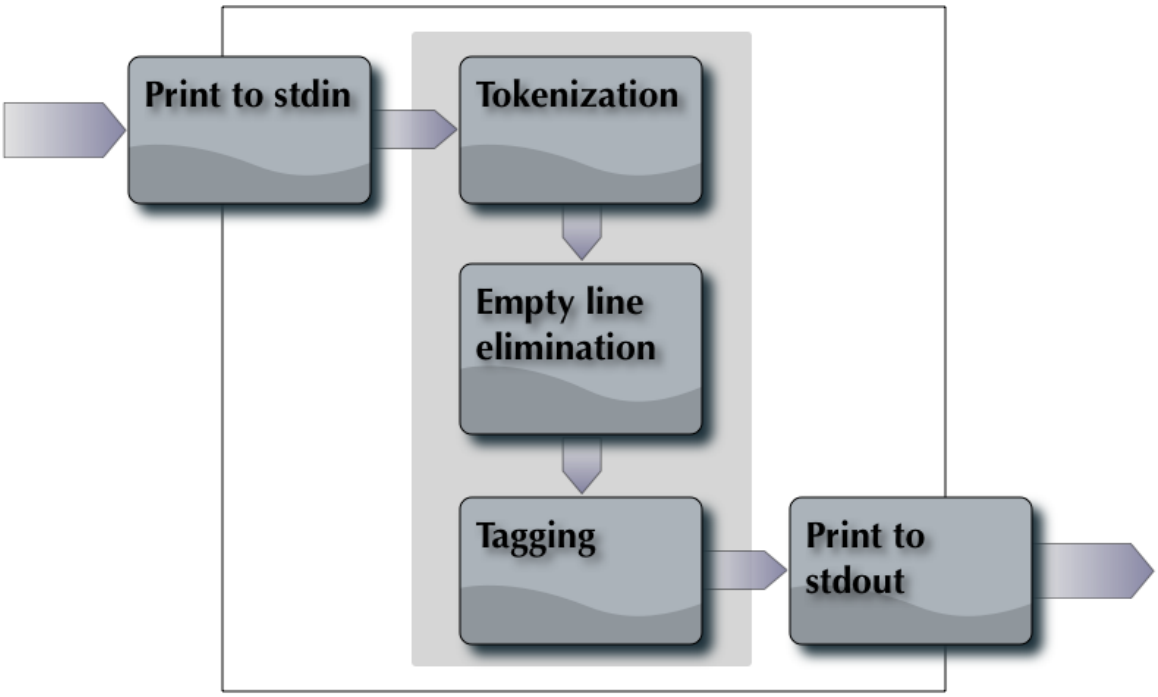

Figure 5.9.: Workflow of tokenization and tagging

Before the tagging can take place, tokenization needs to be done, i.e. the corpus is split into meaningful tokens. This step is necessary before any tagger can start its tagging. Sometimes a tagger can tokenize the input text itself but we want to do this step before any tagger comes into action, because this minimizes the differences in interpretation of tokens in case of more than one tagger used. If we do the tokenization for all taggers the same way, the problem of different interpretations is solved.

After the tokenization, all empty lines in the text are eliminated. Then the texts are passed to the tagger for tagging. The behavior of the tagging can be influenced by the user. Different taggers are available. Which ones are available depends on the specified language. In general, two different tagging variants are possible:

Combined tagging All available taggers for the specified language are used to tag the text. After that, their results are combined using the simple voting algorithm. Figure 5.10 shows this combination. The advantage of combined tagging is a higher accuracy rate, but unfortunately this needs more execution time than the use of one single tagger

only. The usefulness of a combination depends on the language. The combination option is chosen as default if nothing else is set.

Single tagger It is possible to set explicitly one tagger, which skips the use of the other taggers and thus saves processing time. This method is only as accurate as the chosen single tagger itself, but the advantage is higher execution speed.

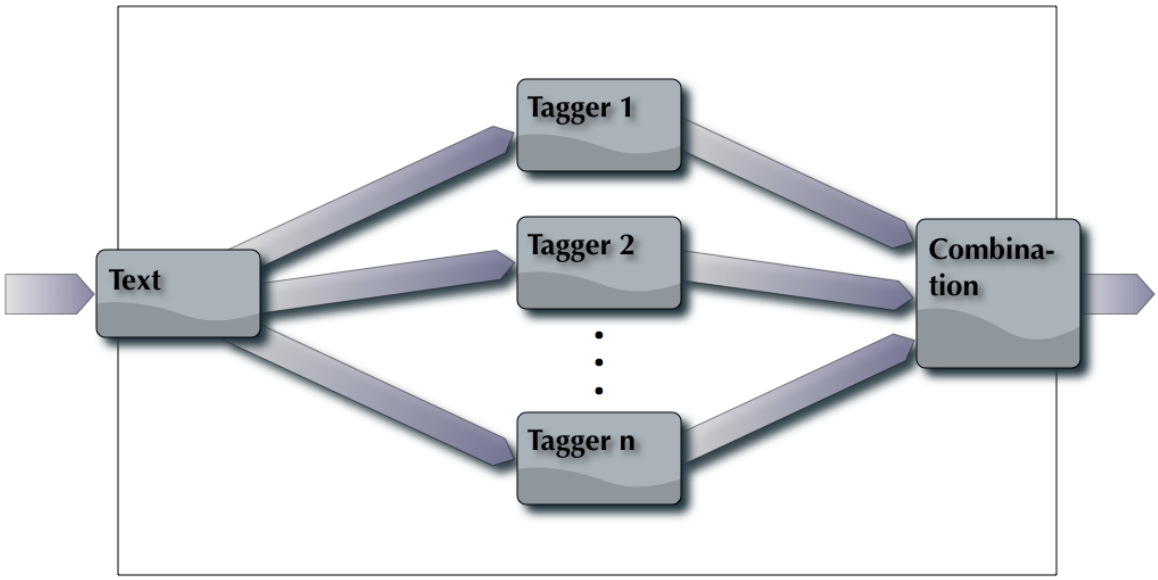

Figure 5.10.: Tagger combination

All taggers are programs that are called within LISGrammarChecker. Because of that, Figure 5.9 illustrates an initial step of printing text to stdin. This means, that LISGrammarChecker redirects all texts to the standard input of the taggers, i.e. to the tokenization components of the taggers. The output of the taggers are again standard streams. These streams are directly used by LISGrammarChecker.

Afterwards, the tagged texts are passed to the next components (data gathering component in training mode and checking methods component in checking mode).

5.6. Data

We use a MySQL [MS] database system to store the data. The database is used through the C bindings. We plan to implement an interface that communicates with the database. This interface can be used by all functionalities to read data from database or to write data into

the database. This design offers encapsulation insofar that not every program element can access the database. The database is rather restricted to one component which manages all database accesses.

Table 5.3 shows the data which is stored in the database. The third column shows that every type of data are stored in an individual database table. These database tables are shown in Figure 5.11. In this figure, the token n-grams from Internet search are missing. The reason is a better manner of representation of the picture. The Internet n-grams would be drawn exactly the same as the token n-grams. This means that four more tables (Internet penta- to bigrams) exist.

Table 5.3.: Data in the database

Checking method	Type of data	Database table
Token n-gram check (Internet functionality irrelevant)	Token bigrams	2_GRAMS
	Token trigrams	3_GRAMS
	Token quadgrams	4_GRAMS
	Token pentagrams	5_GRAMS
Token n-gram check (Internet extension activated)	Token bigrams from Internet	2_GRAMS_INTERNET
	Token trigrams from Internet	3_GRAMS_INTERNET
	Token quadgrams from Internet	4_GRAMS_INTERNET
	Token pentagrams from Internet	5_GRAMS_INTERNET
Tag n-gram check	n-gram of all tags of a sentence	SENTENCE_TAGS
	Tag bigrams	2_GRAMS_TAGS
	Tag trigrams	3_GRAMS_TAGS
	Tag quadgrams	4_GRAMS_TAGS
	Tag pentagrams	5_GRAMS_TAGS
	Bigrams of a token with its left neighbors tag	TAG_WORD
	Bigrams of a tokens with their right neighbors tag	WORD_TAG
	Trigram of a tokens with its left and right neighbors tags	TAG_WORD_TAG
Adverb-verb-agreement	Adverb tokens with belonging verb tags	ADVERBS_VERSB
Adjective-noun-agreement	Adjectives and belonging nouns, both as tokens	ADJECTIVES_NOUNS

Figure 5.11 illustrates that every type of data have an associated field for the amount. This amount represents the occurrences of an n-gram in training mode, or in case of the Internet n-grams, the amount of e.g. Google [Gooa] results is stored.

There are a few details, which are not mentioned until now, e.g. every table has an id field and there two more database tables WORDS and TAGSET. These serve as ancillary tables to facilitate the use of IDs instead of the tokens or tags. This database design results from optimization steps which amongst others comes from database normalization. The database

is normalized in third form—aside from table SENTENCE_TAGS. This table is intentionally in alternative format. Normalized in third normal form means that the database satisfies the first three normal forms as follows [Hil]:

First normal form A database satisfies the first normal form, if all tables are atomic. This means that each value in each table column is atomic, i.e. there are no value sets within a column.

Second normal form The basis for the second normal form requires a database to satisfy first normal form. Additionally, every non-key column must depend on the entire primary key.

Third normal form The third normal form bases on the second normal form. Furthermore, every attribute that is not part of the primary key depends non-transitively on every key of the table. Simply said, this means that all columns directly depend on the primary key.

LISGrammarChecker attempts to be language independent, thus the database model needs to serve several natural languages. The database tables from Figure 5.11 do not include any possibilities to mark the language of the entries. If we consider that all tables contain a huge amount of data from just one language, the data retrieval for more than one language in the same table will be heartaching slow. We want to solve this issue by using a new set of database tables for each language. Thus all database tables in Figure 5.11 are exclusively available for the language that is currently used. The database interface knows the current language with which LISGrammarChecker runs. Thus for the first time that a language is chosen, all database tables are created for this language. If the database tables for the current language already exist, these are simply used—data is read from the corresponding database and written to the correct one. Therefore there is a set of database tables for every language that has already been used in LISGrammarChecker.

Furthermore, there is the possibility to specify a database name with an input parameter. In this case, LISGrammarChecker uses not the default database name, but the new one. If the database with the specified name does not yet exist, it creates a completely new database and tables for every natural language that is used with this database. This possibility can be useful if another training corpus should be used, which is not intended to be mixed with an already existing one in the same language.

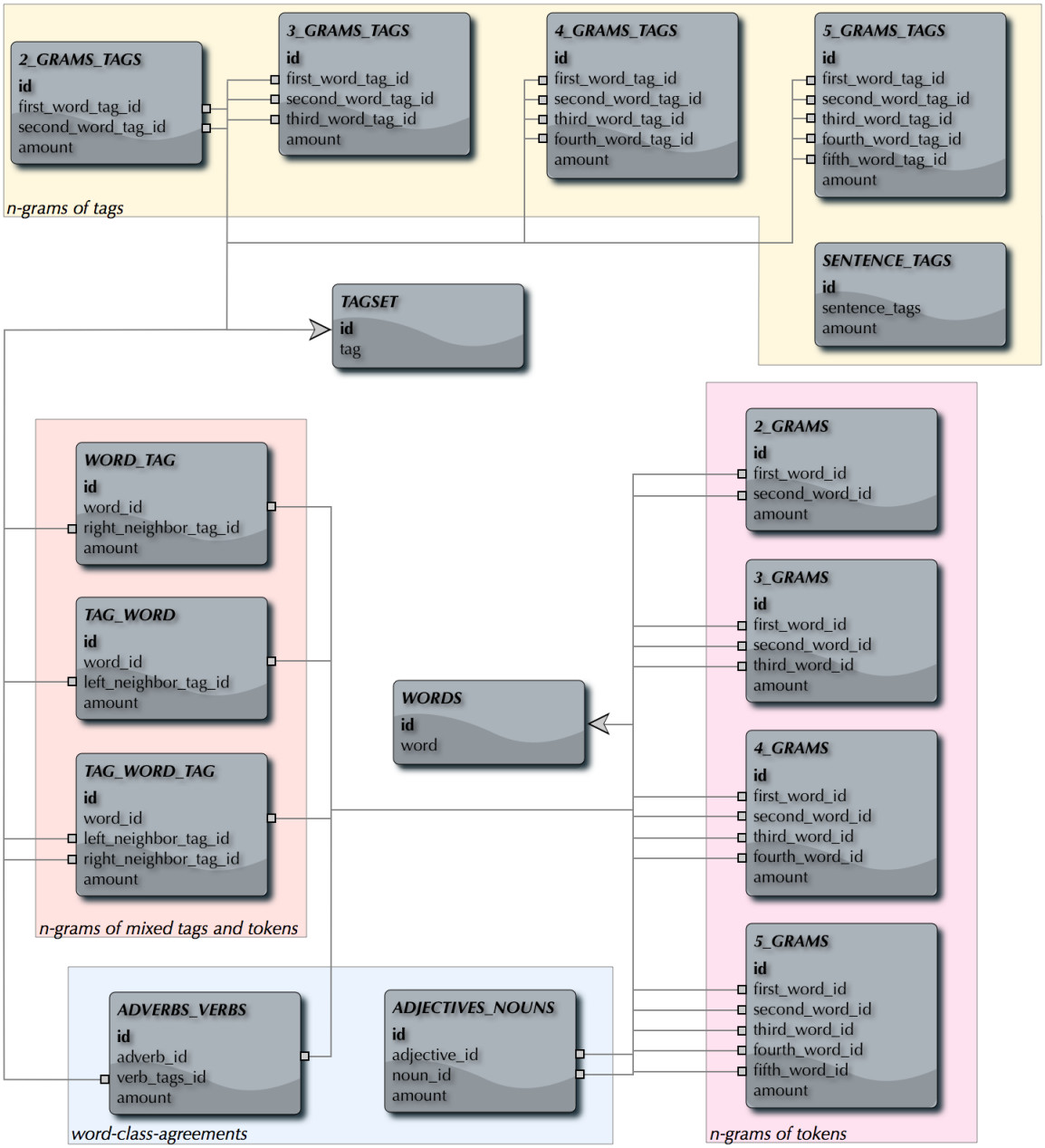

Figure 5.11.: Database structure with tables

6 Implementation

Here we provide an overview of the implementation of LISGrammarChecker. The program is mainly written in D but it also uses some help of a few shell scripts. We start with the description of the user interface. Then we explain how we have realized the tagging. As the main functionality of LISGrammarChecker is split up in two different modes—training and checking—which represent different fields of activity, we explain them separately. Finally, everything concerning data is described.

6.1. User Interaction

LISGrammarChecker has no graphical user interface. It is controlled by command line switches similar to other POSIX [IEE04] programs. These switches are given to the program as arguments. This facilitates the use of data preprocessing tools (like charset converters) before the data is delivered to our grammar checker.

On execution of the program without parameters, a help screen is shown (see Listing 6.1). This screen explains the usage of LISGrammarChecker. It tells e.g. how to specify the language or the tagging options. The usage provides information about the relevance of the

parameter—if these are required or optional. In case of an optional argument, default values are mentioned. There are a few parameters that can be used in both modes, e.g. database name or tagging mode. But some are only relevant in one mode, e.g. the error threshold only makes sense in checking mode. At the bottom of Listing 6.1, there are two examples how to call LISGrammarChecker.

```
LISGrammarChecker -- Language Independent Statistical Grammar Checker
2009-01-31, by Verena Henrich and Timo Reuter

Usage: ./LISGrammarChecker.out [OPTIONS] FILE

Options:
    -d DBNAME,
        --dbname DBNAME  Use database with name DBNAME. This value is optional,
                         default is "lis_grammar_checker".
    -D, --droptables     Drop all tables from the database.
    -e NUM,
        --threshold NUM  Set error threshold value, standard is 30 (ignored in
                         training mode).
    -h  --help           Print this help.
    -l LANG,
        --language LANG  Select language. Options are "de", "en" (default)
                         and "is".
    -s [NUM]
        --search [NUM]   Use an Internet search engine (i.e. Google) for
                         additional analysis (ignored in training mode). NUM
                         specifies threshold for amount of search results that
                         defines positive search. NUM is optional, 400 is default.
    -T  --training       Use training mode instead of checking mode (default).
    -t TAGGER,
        --tagger TAGGER  Select tagger. Option "all" (default) uses
                         simple voting results of all taggers, other options
                         are "tnt", "treetagger" and "stanford".
    -v  --verbose        Activates further information. In training mode, e.g.
                         information about gathered n-grams, in checking mode,
                         e.g. results of every n-gram check request.

Examples: ./LISGrammarChecker.out -T -D --language de statistical.text
          ./LISGrammarChecker.out -t tnt -s --threshold 15 input.text
```

Listing 6.1: Program usage information

The parameters are given as arguments to LISGrammarChecker. Thus, they can be accessed through the `args[]` array in the main routine. We retrieve the given parameters as Listing 6.2 rudimentally illustrates. This implementation allows any order of the input options. Dependent on the specified arguments, we trigger the corresponding events. The listing shows a switch statement, where a string `args[i]` is used in the switch expression. This is one positive aspect of the programming language D which in this case leads to a more straightforward code.

```
for (int i = 1; i < args.length; i++)
{
    switch (args[i])
    {
        case "-D":
        case "--droptables":
```

```
7               droptables = true;
8               break;
9
10          case "-T":
11          case "--training":
12              training = true;
13              break;
14
15          case ...
16      }
17  }
```

Listing 6.2: Retrieval of the input parameters

6.2. Tokenization

Before any tagger can start its work, tokenization must be done as a preprocessing step. Therefore, we use a Perl script, called `tokenize.pl`[1]. The script works as described in the theory in section 4.5. It performs several tasks to achieve the overall goal of tokenization and counteracts corresponding problems. Mainly, the input texts are split into tokens—one token per line. Thereby, periods are disambiguated, and e.g. clitics, punctuation, and parentheses are considered as own token. The script should not get too much text as input, because it reads the whole input file at once and needs therefore a lot of memory.

The tokenization is included as a preprocessing step in the tagging component. This is described in the following section.

6.3. Tagging

In LISGrammarChecker, there are two different tagging methods possible: combined tagging or the use of one single tagger. At the moment, we support three different taggers—TnT [Bra00], TreeTagger [Sch94] and Stanford Tagger [TM00]. Combined tagging with all available taggers for a language is the default behavior. If it is desired to use a single tagger, this can be done by the input parameter, e.g. `--tagger tnt` to use TnT.

The main tagging module is called `taggers.tagger`. The function `runTaggers` manages tagging, regardless of the chosen language or tagging variant. All used taggers are executed through direct calls from inside LISGrammarChecker. No extra program needs to be called beforehand. We can call external programs with the help of our `shell` function in module

[1]The script we use is written by Helmut Schmid, Institute for Natural Language Processing at University of Stuttgart, Germany (http://www.ims.uni-stuttgart.de/ftp/pub/corpora/tokenize.perl)

standard.shell. The working of this function is described in section 6.4. Hereby, we can retrieve the stdout of the called program for further processing.

Every included tagger is represented by a module inside taggers, e.g. taggers.tnt or taggers.treetagger. Using the shell function, we call an individual shell command for each tagger. These commands contain details about all parts from the grey box in Figure 5.9: tokenization, empty line elimination, and tagging. Listing 6.3 shows an example tagger call. The command illustrates a call of TnT tagger in its German version.

```
1 // Call TreeTagger with German parameters
2 shell("echo '" ~ inputtext ~ "' | ./taggers/tokenize.pl -a taggers/abbreviations/german |
      grep -v '^$' | ./taggers/tnt/tnt -v0 ./taggers/tnt/models/negra -");
```

Listing 6.3: Example command to call tagger

The first part of the command (echo) prints the input text. This text is piped (|) into the perl script (./taggers/tokenize.pl). Here, piping means that the stdout of the left part of the pipe symbol (|) serves as stdin for the right part of that symbol (see first box in Figure 5.9). This means that the printed input text serves as input to the perl script. This script does the tokenization (second box in the figure) as described in the previous section. The next step eliminates all empty lines (third box in Figure 5.9). This is done with the command grep -v '$'. The result is a list of tokens separated by carriage returns (\n). This result is piped into the tagger program itself (./taggers/tnt/tnt). One speciality in this command to call TnT is the hyphen (-) at the end. This enables us to give TnT a piped stdin as input which simulates a file. This is necessary because TnT does not support reading input text from stdin.

The commands for other languages and taggers are similar. The outputs of all taggers vary. We harmonize them and store all together in evaluated_lexemes_lemmas_tags, a three-dimensional char array. This array represents the tagging result in case of combined tagging as well as a single tagger is used. To describe the content of this array, let us think about a two-dimensional table, where all cells represent strings, i.e. one-dimensional char arrays. An example is shown in Table 6.1. The first column contains the unchanged tokens (lexemes) of the input text. The second column contains the base forms (lemmas) of these tokens. The third column represents the tags of the final tagging result. Further columns contain the tags of the individual single taggers that are used.

Regardless of the used tagging variant, the tagging result is stored in the third column of evaluated_lexemes_lemmas_tags. In case one uses just a single tagger, this array has only three columns, and the third one contains the result of the single tagger. Otherwise, if combined tagging is used, at first all single taggers are executed. Their proposed tags are stored into columns four onwards, leaving column three empty for the final combined result. The next step is the combination of all proposed tags. Since the final result is stored in the

Table 6.1.: Content of array evaluated_lexemes_lemmas_tags

Lexemes	Lemmas	Result	Tagger 1	Tagger 2	Tagger 3
This	this	DET	DET	DET	DET
shows	show	VBZ	VBZ	NNP	VBZ
a	a	DET	DET	DET	DET
house	house	NN	NN	NN	NN
.	.	SENT	SENT	SENT	SENT

same way regardless of the used tagging variant, further processing is not influenced if a different tagging method is used.

In LISGrammarChecker the simple voting combination algorithm is implemented. The tag which is proposed by most of the taggers is used as the final annotation. In case of a tie—when using three taggers, this can occur if all tagger propose a different tag for a certain token—the tag of the first tagger is used. This is the tag proposed by TreeTagger and this behavior can only be changed if the execution sequence is reordered. Listing E.1 in the appendix shows function `simpleVoting` with the implementation of the simple voting algorithm. The algorithm goes through all tags. If it finds a new tag, a new entry with amount one is created. If the tag is already in the list, the amount is increased. As final step, the amounts are compared and the tag with the highest amount is returned.

Currently, LISGrammarChecker includes three taggers. But nevertheless the algorithm is designed to take a quasi-unlimited amount of tagger outputs to combine them to one result.

The demand to add further taggers could arise, especially to facilitate language independence. A new tagger need to be pretrained in order to be able to tag. New taggers can simply be added by copying all needed tagger resources into the `taggers` module and providing a module `taggers.newtagger`. This module can use the `shell` function to call the new tagger. The output on stdout is returned to LISGrammarChecker which if necessary needs to be adapted in order to fit into `evaluated_lexemes_lemmas_tags` array.

6.4. External Program Calls

As already described in section 4.3, the D programming language has the capability to execute an external program. Unfortunately, it is impossible to fetch the output of that program

from the standard input (stdin). We tackle the problem introducing a new function called `shell` (shown in Figure 6.1) in module `standard.shell`. It works similar to the function which will be provided in version 2.0 of D. Our function links directly to the `popen` function from C which provides a method to redirect the stdout. This gives us the capability to store the output of a program in a variable. The full code is shown in Listing E.2 in the appendix.

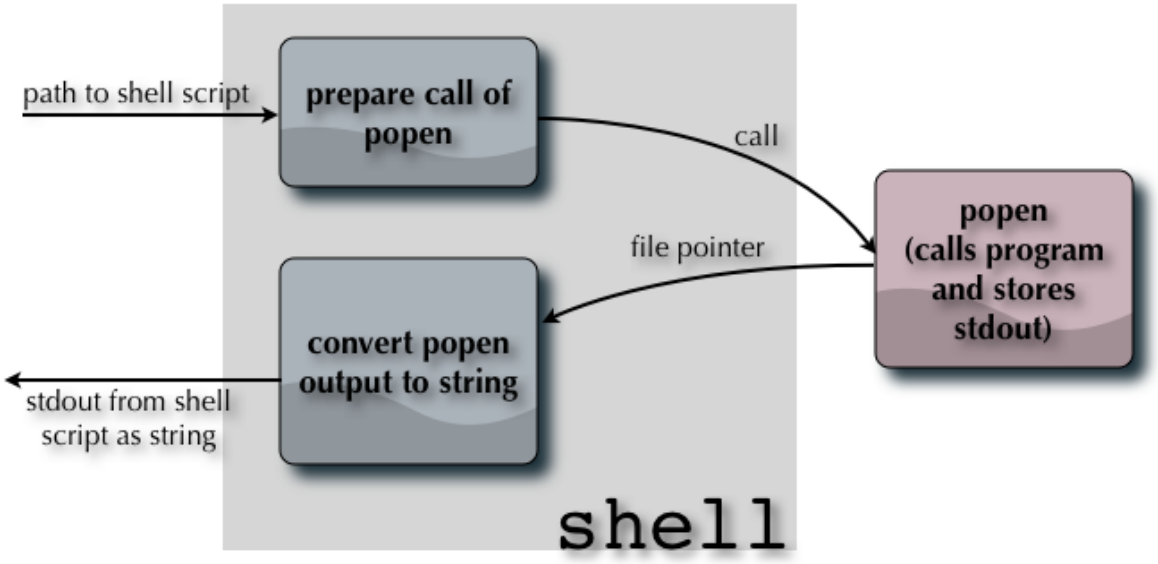

Figure 6.1.: Schema of shell function

6.5. Training Mode

The main task in training mode is gathering statistical information. Prior to the data gathering component, the input text gets tagged. Then, several properties of the input texts are analyzed and needed information is extracted and finally stored to the database. This mode is activated with the input parameter `--training`.

The data gathering task is mainly done in three functions: one for the extraction of n-grams (both tokens and tags), a second for the extraction of the adverbs and verb tags, and a third for the adjectives and nouns. These functions correspond to the main grammar checking features of our program.

Extract n-grams The handling of the n-grams is inside the module `standard.neighbors`. Here, the function `evaluateNeighbors` implements the data gathering of both the n-grams of tokens and the n-grams of tags. This speeds up the execution speed of the

program. The algorithm doing the extraction goes through all sentences and in every step it performs the following:

1. Add tags of the whole sentence to the database.
2. Regard every word of the sentence and determine all bi-, tri-, quad-, and pentagrams of tokens, all bi-, tri-, quad-, and pentagrams of tags and all hybrid bigrams and trigrams for each token.
3. Save all n-grams to the appropriate database tables.

Extract adverbs and verbs In module `standard.adverbs` the adverb-verb-agreement is implemented. Function `evaluateAdverbs` extracts all adverbs and the corresponding verbs. The text is analyzed sentence by sentence. In every sentence the occurrence of an adverb is checked. This is done with a check, if one tag in the sentence marks an adverb. A call of function `bool isAdverbTag(char[] language, char[] tag)` gives this information as a boolean value that marks the `tag` in the `language` as an adverb. This function is not language independent and needs to be extended if not yet implemented languages should be used. In case of a found adverb, the token of the adverb is stored, and additionally all verb tags of that sentence are extracted. The verb tags are detected with the help of function `isVerbTag`, which works similar to `isAdverbTag`. This information (the adverb tokens and the verb tags) is written to the database.

Extract adjectives and nouns The module `standard.adjectives` includes the adjective-noun-agreement. The data gathering is done in function `evaluateAdjectives`. This function goes through all sentences and analyses the occurrence of a noun which is described by an adjective. If these two word classes occur, both the adjective and the noun are stored in the database. The occurrence of the two word classes are gathered through the functions `isAdjectiveTag` and `isNounTag`. These two functions work similarly to the `isAdverbTag` function for adverbs.

6.6. Checking Mode

Checking mode is activated with the input parameter `--checking`. Prior to the checking methods, the input texts are tagged. Then the checking methods analyzes several properties of the input texts and gives back the amount of not found data sets in the database.

6.6.1. Checking Methods

The checking is done sentence by sentence. There are four main checking methods that are executed:

Tag n-gram check This method in implemented in function `analyzeNeighboredTags` of module `standard.neighbors`. It analyzes all tag n-grams. The workflow therefore is shown in Figure 6.2. It starts with checking the sequence of all tags in the sentence. If this sequence is not found in the database, an error is assumed and the next check regards the all tag pentagrams. For every nonexistent pentagram the location, i.e. the token where it occurs, is stored. In case of at least one nonexistent pentagram, the tag quadgrams are checked. Not all quadgrams are checked, but only those which are located inside the wrong pentagram windows. This is realized through the stored pentagram location, and minimizes the amount of quadgram checks so that only the really needed checks are done. For example, if one tag pentagram is false, the two tag quadgrams inside this pentagram window need to be checked. If two neighboring pentagrams are false, three quadgrams need to be checked accordingly. This optimization reaches also to the trigram and bigram checks. The tag trigrams are checked in case of at least one false tag quadgram. The bigrams are only checked in case of at least one missing trigram, and also only these bigrams are checked, which are inside the wrong trigram windows.

All tag n-gram checks which are explained up to now are summarized on the left part of Figure 6.2. Beside these tag n-grams, there are three n-gram checks which regard hybrid n-grams of tags and tokens. The trigram which represents a tag-token-tag sequence is checked at first. Then the two bigrams are checked, which both represent a token and the tag of its neighbor, either the sequence tag-token or token-tag. The checks of these three hybrid n-grams are also optimized, i.e. they are only done when they are really necessary.

Token n-gram check The token n-gram check is also in module `standard.neighbors`. The function doing this work is called `analyzeNGrams`. It analyzes all token n-grams. The workflow is similar to the first part of the tag n-gram check. It is shown in the left side of Figure 6.3. The token n-gram check starts with checking all tag pentagrams of a sentence. For every nonexistent pentagram the location, i.e. the token where it occurs, is stored. In case of at least one nonexistent pentagram, the token quadgrams are checked. Not all quadgrams are checked, but only those, which are located inside the wrong pentagram windows. This works exactly as described above.

The token n-gram check can be extended by n-grams from an Internet search engine. This extension is deactivated by default because of the numerous requests to the Internet. It can be activated by the input parameter `--search NUM`. NUM optionally

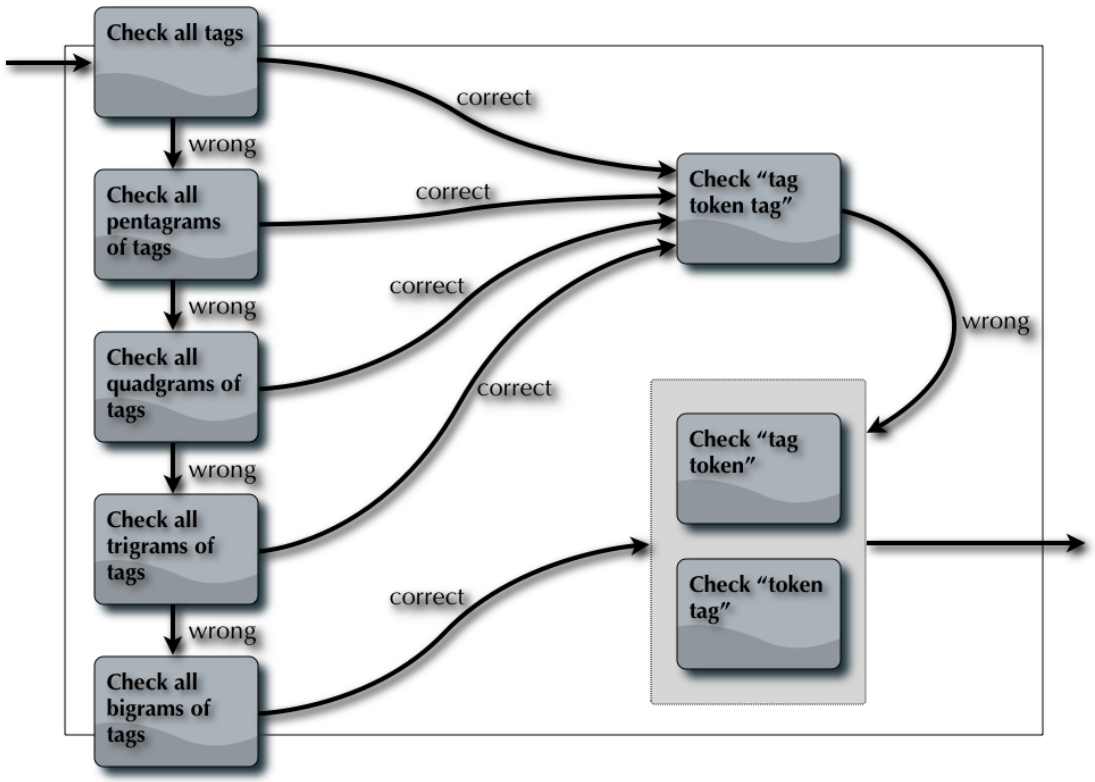

Figure 6.2.: Tag n-gram check in detail

specifies a threshold which defines the amount of search results that defines a positive search. If the Internet functionality is activated, the checking workflow is different. If there is an error in the token pentagrams from our database, the next step is a token pentagram check in the Internet. The Internet pentagram is defined to be correct if the search engine result is higher than the specified threshold (default is 400). If this pentagram is not correct, then we continue with the token quadgram check from the local storage, and so on. The new workflow is shown on the right side of Figure 6.3. For every Internet search request that is done, we save the result. Thus if a request was already done, the result can be taken from the database and an unnecessary Internet request can be avoided. Further details about how we establish a connection or how we read data from the Internet are explained in the next subsection.

Adverb-verb-agreement This functionality is in function `analyzeAdverbs` which is situated in module `standard.adverbs`. It works similar to the corresponding function in training mode. All sentences are analyzed with regard to an adverb occurrence. This is again done with the help of function `isAdverbTag`. In case there is an adverb in the sentence, all verb are searched with the help of function `isVerbTag`. The combination

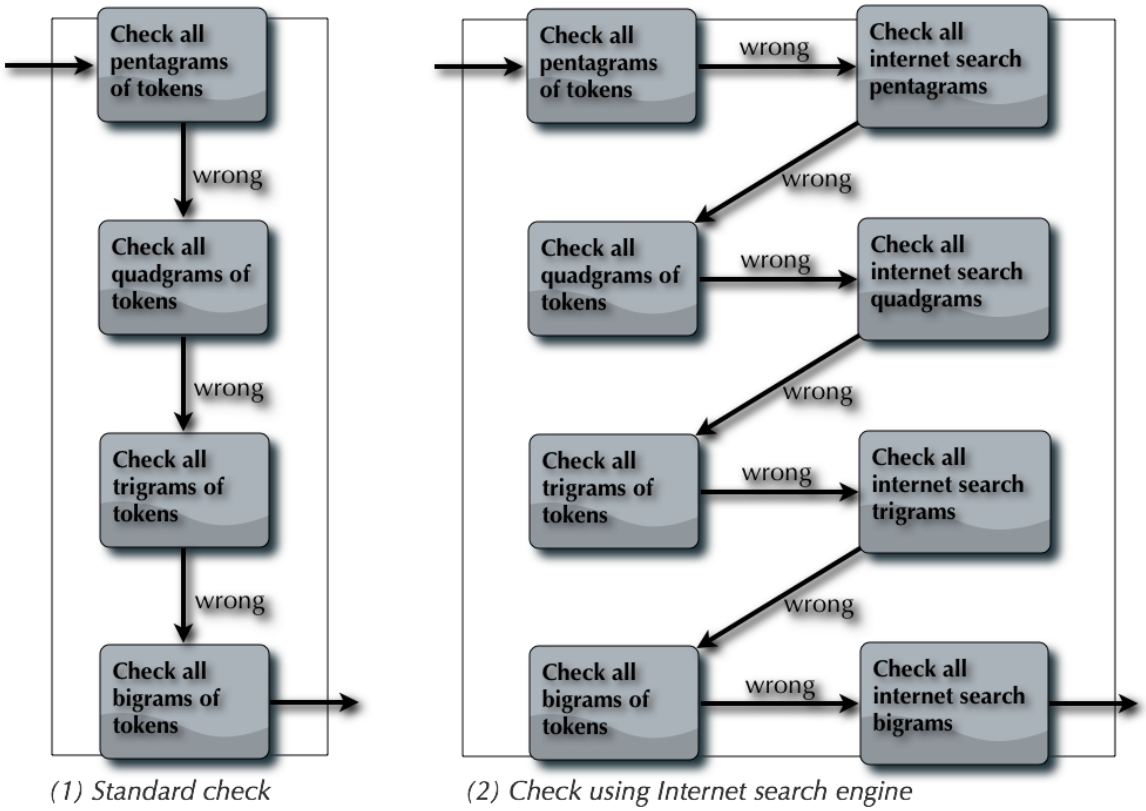

Figure 6.3.: Token n-gram check

of the adverb token and all verb tags of the sentence are checked in the database. If the combination is nonexistent, an error is marked.

Adjective-noun-agreement Module `standard.adjectives` contains the responsible function `analyzeAdjectives`. This function checks all sentences for an occurrence of a noun (`isNounTag`) with a describing adjective (`isAdjectiveTag`). This combination is then looked up in the database. In case of a negative result, an error is marked also.

6.6.2. Internet Functionality

This functionality is implemented in module `standard.search`. We have implemented the Internet search engines Google [Gooa], Google Scholar [Goob] and Yahoo [Yah]. Google is used as the standard at the moment.

Inside the module, the function `getDataFromConnection` establishes an Internet connection using an Internet address (see Listing 6.4). The command `writeString` is used to send

an HTTP request to the server. The answer is read with `socketstream.readLine()` and contains an HTTP header with HTML code.

```
// Create new TCP socket and socket stream from given parameters,
// InternetAddress resolves domain to IP address
auto Socket tcpsocket = new TcpSocket(new InternetAddress(domain, port));
Stream socketstream = new SocketStream(tcpsocket);
...
// Send a GET request to the server
socketstream.writeString("GET " ~ url ~ " HTTP/1.1\r\nHost: " ~ domain ~ "\r\n\r\n");
```

Listing 6.4: Establish Internet connection

The `url` in the string to the server (see Listing 6.4) is different for all searching machines. For Google it is `http://www.google.com/search?ie=utf-8&oe=utf-8&q=` for example. After the last equal sign there needs to be the string which should be searched. It needs to be given in double quotes and all white space characters must be replaced by the plus sign. If we send modern houses are as request, the string to the server is the following

```
http://www.google.com/search?ie=utf-8\&oe=utf-8\&q=''modern+houses+are''
```

Google gives back its standard result page. We do not send the request via browser and get thus the plain HTML code. The HTML data is parsed to extract the amount of search results. In case of Google the phrase swrnum=123456 is extracted from the data stream. The value after swrnum= can be used directly as the amount of results. Listing 6.5 illustrates this.

```
// Find variable "swrnum", which shows the amount of results
int pos_amount = find(html_data, "swrnum") + 7;
// Count the digits in the string
auto digits = countchars(html_data[pos_amount..pos_amount+9], "0123456789");
// Get the amount of results and convert it to integer
amount_of_google_results = toInt(html_data[pos_amount..pos_amount+digits]);
```

Listing 6.5: Gain Google search result amount

The same is done with Google Scholar and Yahoo.

6.6.3. Correction Proposal

Function `proposeCorrection` gets the erroneous pointed token phrase as an argument. We check all tri- and pentagrams of tokens with functions `getCorrectionProposalFrom3Grams` and `getCorrectionProposalFrom5Grams`. These functions return the most probable token for the position in the middle. Thus we check the trigram sequence token * token with the wildcard at the second position and the pentagram sequence token token * token token with the wildcard at the third position.

6.6.4. Grammar Checking Output

Listing 6.6 shows an example output of a grammar check. The command to call LISGrammarChecker in this specific example is:

```
./LISGrammarChecker --tagger tnt --language en --threshold 5 input.text
```

The listing shows that LISGrammarChecker finds the error and points it to the user. It does not show every detail—the option `--verbose` can extend the output—but the relevant parts to get a feeling for the output.

```
Run LISGrammarChecker in language en
    - Use TnT Tagger
    - Use error-threshold 5


########## Tag n-gram check ##########

Analyze tag n-grams in sentence "These (DD2) also (RR) find (VV0) favor (NN1) with (IW)
     girls (NN2) , (YC) whose (DDQG) believe (VV0) that (CST) Joan (NP1) died (VVD) from (
     II) a (AT1) heart (NN1) attack (NN1) . (SENT)"
1 unknown 3-consecutive-tags combination(s) found:
whose believe that

Overall error counter for tag n-gram check is 6, this is higher than threshold 5.


########## Token n-gram check ##########

Analyze token n-grams in sentence "These also find favor with girls , whose believe that
     Joan died from a heart attack ."
1 unknown 2-gram(s) (not found in the database):
whose believe

Overall error counter for token n-gram check is 10, this is higher than threshold 5.
```

Listing 6.6: Example grammar checking output

6.7. Database

We use a MySQL [MS] database to store all statistical data and other data that are produced during program execution and needs to be stored. In this section we explain how we have implemented the communication with the database and present our database model. Furthermore, we reveal how we got our statistical data and which problems we had to consider.

The input parameter `--dbname DBNAME` switches to the database with name DBNAME instead of the default database `lis_grammar_checker`.

6.7.1. Database Structure/Model

The database model which is used in our approach is described in 5.6. To ensure language independence, we separate all data for each language. We realize this through an exclusive set of database tables for every used language. This means that all tables from Figure 5.11 are available for every language. Therefore, every table is marked with the language code as prefix to its original table name. That means for example the table 2_GRAMS is named EN_2_GRAMS for the English token bigrams; for German this table will be named DE_2_GRAMS.

6.7.2. Communication with the Database

Before using MySQL [MS] with D, the bindings for C must be installed. We use a source file named `mysql.d` which contains the bindings[1] for MySQL. This source file provides an adapter from D to the C bindings of MySQL.

When compiling the program, everything needs to be linked with the libraries pthread, m and mysqlclient so that the database can be used within the program. This requirement results in the following command to compile LISGrammarChecker:

```
1 # gdc *.d -lpthread -lm -lmysqlclient
```

We have a module (`data.mysql`) with the MySQL bindings. Module `data.database` is an abstraction layer between the database itself and the rest of the D program, i.e. a communication interface to the database. It includes functions to establish and close database connections (`establishDatabaseConnection` and `closeDatabaseConnection`), to request data from database (all functions with naming convention `getEntryFromDBNAME`), and to write data into the database (all `addEntryToDBNAME` functions).

The program does not work, if the used database and tables do not exist. The same applies for username and password of the MySQL database. Our database functionality uses the username lisgchecker and password lisgchecker. The user and the database can be created using any MySQL client. The following SQL queries are needed to prepare MySQL for the use with LISGrammarChecker:

```
1 CREATE USER lis_grammar_checker@localhost IDENTIFIED BY 'lis_grammar_checker';
2 CREATE DATABASE lis_grammar_checker;
3 GRANT ALL ON lis_grammar_checker.* TO lis_grammar_checker;
```

The first SQL query adds the standard user. The second creates the standard database and the last query is used to allow the user to access the database with all its tables. If everything is done without errors, the database functionality should work without problems.

[1] The bindings we use are written by Manfred Hansen (http://www.steinmole.de/d).

A database connection in LISGrammarChecker is initialized with the following function calls (excerpt from function `establishDatabaseConnection`):

```
// Initialize MySQL
mysql = mysql_init(null);
...
// Connect to MySQL database
// Parameters: handle, host, username, password, db name, port, unix socket, clientflag
mysql_real_connect(mysql, "localhost", "lis_grammar_checker", "sgchecker", dbname.ptr,
     3310, null, 0);
```

Similar is the call in function `closeDatabaseConnection` to close the connection to the database again:

```
// Close connection to database
mysql_close(mysql);
```

All queries to the database need to be null terminated. While D knows strings as char arrays, it does not contain the null termination known from cstrings. Thus, all strings need to be converted to C-like strings using the D function `toString` or a null termination (\0) must be insert to the string. If this is not done, calls to the database are executed randomly or throw undefined errors.

We have implemented the input parameter `--droptables`. If it is specified, the function `dropTables` is called which drops all database tables. This functionality is contrary to the `createTables` function, where all tables that do not yet exist are created. Both functions can be executed within the same program call, e.g. to setup a new statistical database.

Our approach produces a lot of database transactions. To improve the execution speed when calling the database we have found a way to minimize database queries. Every time we want to save a data set into the database we usually need to know beforehand if the current data already exist or not. One example for that is the table of bigrams. We want all bigrams occur only once in the database. Usually one has to make a query to search in the database to see if the bigrams already exist. If it exists, the counter for it needs to be increased and the entry must be updated. If it does not exist, the entry is added to the database.

```
char[] query = 'SELECT amount FROM 2_grams WHERE first_word=2 AND second_word=7;\0';
mysql_query(mysql, cast(char*)query);

char[] result = mysql_store_result(mysql);

if (char[] bigram_amount_char mysql_fetch_row(result) != null)
{
    int bigram_amount = toInt(bigram_amount_char);
}
else
{
    int bigram_amount = -1;
}

if (bigram_amount == -1)
{
```

```
17      query = 'INSERT INTO 2_grams (first_word_id, second_word_id) VALUES ("2", "7");\0';
18  }
19  else
20  {
21      bigram_amount+=1;
22      query = 'INSERT INTO 2_grams (amount) VALUES ("' ~ bigram_amount ~ '");\0';
23  }
```

We have a quirk to facilitate these queries. We cut the whole process down to one query. To do that, all database table entries need to have those fields defined as unique which are inserted. In case of our example, the bigrams, both the first word and the second word are defined together as unique. If this is done, the query in Listing 6.7 can be used instead of the Listing above.

```
1  char[] query = 'INSERT INTO 2_grams (first_word_id, second_word_id)';
2         query ~= 'VALUES ("2", "7") ON DUPLICATE KEY UPDATE amount=amount+1;\0';
3
4  mysql_query(mysql, cast(char*)query);
```

Listing 6.7: Complicate SQL syntax to save queries

The query is trying to insert the bigram to the database. If it cannot be written, the command after the ON DUPLICATE keyword is executed. Here the amount will be updated.

Part III.

Evaluation

7 Test Cases

In this chapter, we test LISGrammarChecker in different ways. First, we establish test criteria. We specify which statistical data we use for training and which input data we use for grammar checking. We describe tools which we use for automatic training, grammar checks, and evaluations. Then we show examples of how our program works with various kinds of input texts. We test different languages to show the program's language independence. The examination of large corpora shows the capabilities of our approach in a real environment. Finally, we measure the execution time of LISGrammarChecker.

7.1. Criteria for Testing

All tests are done on a standard computer with an Intel Core Duo processor at 2,0 GHz and 2 GiB of memory. The program runs on a Linux operation system with Kernel 2.6.27. As I/O scheduler we use the CFQ scheduler. The used database system is MySQL [MS] in version 5.0.67. We use UTF-8 encoding [Yer03] for everything, e.g. the database or the input data. All used data—the statistical data for training and the input data which is checked for errors—are stored in individual UTF-8-encoded text files.

7.1.1. Statistical Training Data

To test LISGrammarChecker, we first need a lot of good quality statistical data for training to built up a representative database. Therefore, we use large corpora. These corpora need to be of good quality, i.e. grammatically correct, of good language style, and containing only complete sentences. We use the following statistical corpus data to train LISGrammarChecker:

Wortschatz Universität Leipzig (English) We use this free corpus from Universität Leipzig [QH06] which contains 1 million randomly chosen English sentences. Its sources are AP (years 1988 and 1989), Financial Times (years 1991 to 1994), OTS newsticker and Wall Street Journal (years 1987 to 1992). This collection was build in 2006. The average length of a sentence is 21.1915 words and thus the corpus consists of about 21.2 million words.

Refined Wortschatz Universität Leipzig (English) We use a refined version of this corpus. The main reason for the refinement is the large amount of incorrect data. We hand-corrected the corpus by eliminating all double quotes and to a large extent the single quotes. We deleted meaningless lines and replaced erroneous characters, e.g. erroneously used French accents by apostrophes. Furthermore, we replaced each period at the line end by exclamation points (this avoids confusions with abbreviations at the line end). This refined corpus should improve the statistical database of LISGrammarChecker. This (new) corpus contains 819,210 sentences instead of one million as before.

Wortschatz Universität Leipzig (German) The German corpus from Universität Leipzig is also free. It consists of 1 million sentences from many German newspapers. Among these are TAZ, Die Welt, Die Zeit, Süddeutsche Zeitung, and Frankfurter Rundschau. Furthermore, there are some sentences from online sources like Spiegel Online and Netzzeitung. The corpus was also build in 2006. It contains about 15.7 million words with an average length of 15.7217 words per sentence.

Self-made composition (English) This corpus consists of several sources like e.g. parts from ANC, newspaper texts, and texts from a portal to learn English. We have hand-chosen all sources, and hand-corrected them to avoid incompatible characters during tokenization. We use an extraction from the American National Corpus [IS06]. This part of our self-made composition corpus consists of about 80,000 words from letters and technicals papers. The newspaper texts are composed from Bloomberg [Blo], ABC news [ABC], New York Times [The], and VOA news [VOA], altogether about 20,000 words. Texts from an online portal to learn English [Pöh] contribute about 10,000 words. Thus, this corpus contains about 110,000 words overall.

7.1.2. Input Data for Checking

To perform the proper grammar checking, we use several error corpora. These are all in the following format:

- Every line starts with a letter, either `A`, `B` or `C`. Letter `A` marks intentionally wrong sentences. Letter `B` marks the corrected version of the same sentence. In some cases there exists also a variant `C`, which denotes a second correct alternative of the same sentence.
- A period and a space follow to distinguish between the type marker and the sentence itself.
- Finally the line contains the sentence itself.

In our test cases we use the following input data to check it for errors:

Self-made error corpus (English) We constructed this error corpus on our own. It includes parts from the Wortschatz Universität Leipzig corpus [QH06]. We have randomly selected sentence parts from the corpus and formed new sentences out of them. Now it consists of newly created English sentences. First the sentences are written in a correct form. Later, we have inserted various types of grammatical errors. All grammatical errors from chapter 2.1.4 occur at least once in the corpus. Finally, this corpus contains 264 sentences—131 intentionally wrong sentences and their correct versions (see Listing E.3).

Self-made error corpus with simple sentences (English) We provide a small error corpus with just simple sentences. These sentences are made up by ourselves on the basis of the training corpus denoted as self-made composition in the previous subsection. This error corpus contains 100 sentences, both 50 correct and incorrect ones. It is shown in Listing E.4 in the appendix.

Self-made error corpus (German) This corpus is similar to the self-made error corpus for English. It includes 260 sentences of German text which we made up from different sentence parts out of the Universität Leipzig Wortschatz corpus for German. It includes 130 correct sentences and 130 incorrect ones. This error corpus can be found in Listing E.5.

7.1.3. Auxiliary Tools

To test LISGrammarChecker, both preparatory and subsequent work are necessary. This work is multifaceted, e.g. train the database automatically, do grammar checking, or evaluate test results. We use several self-written shell scripts to perform this work. Because of some

quirks in the used corpora, each needs a different shell script. This is important to ensure that the format of the data is compatible with the one LISGrammarChecker expects as input. Below we show how the different shell scripts work. If a shell script is executed without parameters, it prints out a usage description and an example how to use it.

Train Wortschatz Universität Leipzig (English and German) The corpus from Universität Leipzig is one of the most important ones. Every line contains exactly one sentence and all lines are numbered. Because of some restrictions with the tokenizer (see section 6.2), it is not possible to give the corpus text file all at once to the tokenizer. Instead, the corpus needs to be split into parts with a maximum of about 2,000 words per part. The script for this task splits the corpus after each 100 lines and deletes the numbering at the beginning of each sentence. This is done for the whole corpus. After each 100 lines, this part is trained to LISGrammarChecker. We have implemented one shell script which handles both English and German. This script takes six inputs: a text file to train, a database name where the data should be written to, a tagging method to use, the language (`en` or `de`), a file where the log should be written to, and a number which specifies the first line to be checked. The last argument enables it to skip some sentences. This is useful to pause the training and continue later.

Check error corpora (English and German) Another shell script performs the checks of the sentences from the error corpora. This script can handle every error corpus which is in the specified format that each line starts with a letter `A`, `B` or `C` which classifies a sentence as correct or wrong, followed by a period, an empty space, and finally the sentence itself. The following steps are done in the script to check an error corpus:

1. The first character of each line is extracted by a regular expression to determine if the sentence is correct (character `B` or `C`) or incorrect (character `A`). This information is also used to print out the type of the sentence.
2. The first three characters of each line—letter, period, and empty space—are skipped using a regular expression.
3. The sentence itself is passed to LISGrammarChecker.

These steps are repeated until all sentences are processed. The results are written to a log file. The arguments are the following: a text input file (error corpus), a database name, a tagger variant, and a log file.

Evaluate checking results This scripts uses a lot of regular expressions to parse the log file from the previous script, which contains the results of the checking process.

The checking log file is passed to a concatenated bunch of text search and replace operations. The final result is comma separated data, written to a temporary file. Each line contains the data of exactly one sentence. It contains the sentence number, the

information if the sentence is correct or not (A, B or C), the amounts of not found tag n-grams, hybrid n-grams and token n-grams. These amounts are extracted from the comma separated values and are summed up for all sentences. In the meanwhile we count the amount of sentences. When the processing has terminated, the average amount of not found n-grams per sentence is calculated using the following formula:

$$\frac{\sum \text{not found n-grams}}{\sum \text{sentences}} = \text{average amount of not found n-grams per sentence}$$

First this is done for the tag n-grams and the hybrid n-grams, then it is also done for the token n-grams. The mechanism is done individually for the correct and the incorrect sentences.

Another pattern match counts all different amounts for the individual error types. Here the amount of not found tag pentagrams, tag quadgram, etc. are shown. The script sums up the amounts of all sentences, not separated by correct or incorrect sentences.

Advanced evaluation of checking results This script is very similar to the previous one. It does most tasks analogously but differs in the last step, the counting of all different amounts for the individual error types. The previous script sums up the amounts of all sentences regardless their type. This advanced script differentiates between correct and incorrect sentences and thus sums up the error amounts separated for the type of sentence. Both scripts are necessary because the extended script is very time consuming.

Simplify the evaluation of checking results Some other scripts are used for minor tasks to simplify the evaluation of LISGrammarChecker:

Get corresponding token n-gram for tag n-gram We have written several scripts to retrieve example token n-grams from a corpus for a corresponding tag n-gram. That means that the script takes a tag n-gram or hybrid n-gram as input and returns an appropriate token n-gram for it.

Show unknown words Another script shows all tokens in the error corpus which are not found in the trained statistical data.

Show false positives There is a script which shows all false positives, i.e. the correct sentences which are denoted as wrong by LISGrammarChecker.

7.1.4. PoS Tagger and Tagsets

To run tests with LISGrammarChecker in English, we use either the Penn Treebank tagset [MSM93] or the Brown tagset [FK64]. As tagger we choose TnT [Bra00] for all test cases because of its speed and the highest accuracy of all available taggers (96.7% for the Penn Treebank tagset and 94.5% for the Brown tagset).

To perform German tests, we use the Stuttgart-Tübingen Tagset [STST99]. As tagger we also use TnT with an accuracy rate of 96.7% for the STTS.

7.2. Operate Test Cases

The databases are trained with all the above described statistical data (see subsection 7.1.1). Now we use this data to perform grammar checking. We want to check the texts from subsection 7.1.2. Therefore we define several test cases. All test cases use TnT tagger for tagging. We use error classes A to J to classify the corresponding test results. The error classes are explained in the next chapter (see section 8.2). For most test cases we show separate results for the hybrid n-gram checks in addition to the tag and token n-gram checks. Here we only present the results; the evaluation and interpretation of the results are given in section 8.3.

The values of the tables are determined by some of the scripts described above and by hand.

7.2.1. Case 1: Self-made Error Corpus (English), Penn Treebank Tagset

In our first test case we check our self-made English error corpus. Therefore, we train the database with the English version of Wortschatz Universität Leipzig. We do not use quad- and pentagrams of tokens because we assume that the training speed is a lot faster leaving them out.

All features of our program are tested individually. This means that the tag n-gram check, the hybrid n-gram check and the token n-gram check are treated separately. To get the results, we look at every sentence by hand. Thereby we can see which problems arise in the different parts of the grammar checker and we are able to determine error classes which are described in section 8.2.

The following tables classify the occurring problems to the different error classes. The error classes describe why the grammar checker does not work as expected. This means that

the error classes are the different reasons why LISGrammarChecker erroneously classifies a correct sentence as incorrect or an incorrect sentence as correct.

In the first column of each table the error classes are specified. The third column shows the reasons why LISGrammarChecker classifies an incorrect sentence erroneously as correct. The erroneously marked errors in the correct sentences are shown in the last column—this is also known as the false positive rate. The overall errors that can be classified to an error class—both not found errors in the incorrect sentences and the erroneously as error marked correct sentences—are summarized in the second column.

Table 7.1 shows all these results from the tag n-gram check. The same for the hybrid n-gram check is shown in Table 7.2. The assignment of sentences to the error classes which occur when using the n-grams of token are presented in Table 7.3.

Table 7.1.: Test case 1: Error classification of tag n-gram check result

Error class	All sentences	Wrong sentences	Correct sentences
Too small tagset (A)	15.6% (41)	31.5% (41)	0%
Too little statistical data (B)	4.9% (13)	2.3% (3)	7.5% (10)[1]
Erroneous statistical data (C)	1.9% (5)	3.8% (5)	0%
Tagging error during training (D)	1.5% (4)	3.1% (4)	0%
Tagging error during checking (E)	1.5% (4)	3.1% (4)	0%
Multiple token sequences with same tags (F)	16.4% (43)	33.0% (43)	0%
Sphere of word too large (G)	4.2% (11)	8.5% (11)	0%
Tokenization error (H)	4.6% (12)	5.4% (7)	3.8% (5)
Style or semantic error (I)	3.4% (9)	6.9% (9)	0%
Correct	55.7% (147)	23.8% (31)	85.7% (114)

Table 7.2.: Test case 1: Error classification of hybrid n-gram check result

Error class	All sentences	Wrong sentences	Correct sentences
Too little statistical data (B)	15.9% (42)	14.6% (19)	17.3% (23)

[1] 103 sentences have unknown sentence tags

Table 7.2.: Test case 1: Error classification of hybrid n-gram check result (continued)

Error class	All sentences	Wrong sentences	Correct sentences
Erroneous statistical data (C)	4.6% (12)	9.2% (12)	0%
Tagging error during training (D)	0.8% (2)	1.5% (2)	0%
Tagging error during checking (E)	0.8% (2)	1.5% (2)	0%
Multiple token sequences with same tags (F)	12.9% (34)	25.4% (33)	0.8% (1)
Sphere of word too large (G)	14.8% (39)	13.1% (17)	1.5% (2)
Tokenization error (H)	3.4% (9)	3.1% (4)	3.8% (5)
Style or semantic error (I)	1.1% (3)	2.3% (3)	0%
Check not useful (J)	0.4% (1)	0.8% (1)	0%
Correct	55.1% (145)	32.3% (42)	77.4% (103)

Table 7.3.: Test case 1: Error classification of token n-gram check result

Error class	All sentences	Wrong sentences	Correct sentences
Too little statistical data (B)	58.9% (155)	59.2% (77)	58.6% (78)
Erroneous statistical data (C)	3% (8)	5.4% (7)	0.8% (1)
Multiple token sequences with same tags (F)	0.4% (1)	0.8% (1)	0%
Sphere of word too large (G)	8.3% (22)	15.4% (20)	1.5% (2)
Tokenization error (H)	3.4% (9)	3.1% (4)	3.8% (5)
Style or semantic error (I)	0.8% (2)	1.5% (2)	0%
Correct	55.9% (147)	75.4% (98)	36.8% (49)

If we consider the 131 wrong sentences of the corpus, the following list shows which functionality detects how many errors:

- 28 sentences (21.4%) are not found at all.

- 2 sentences (1.5%) are found by hybrid n-gram check only.
- 48 sentences (36.6%) are found by token n-gram check only.
- 3 sentences (2.3%) are found by tag n-gram check only.
- 21 sentences (16.0%) are found by both, hybrid and token n-gram check.
- 0 sentences (0%) are found by both, tag and hybrid n-gram check.
- 11 sentences (8.4%) are found by both, tag and token n-gram check.
- 18 sentences (13.7%) are found by all three checking methods.

We test the correction proposal for all incorrect sentences where the error is found because it is only useful for these to propose a correction. This means that we look at the proposed corrections of 98 sentences which are 75% of the incorrect sentences. Table 7.4 shows the results.

Table 7.4.: Test case 1: Correction proposal results

Behavior	Rate
correct and expected proposal	10.2% (10)
correct but unexpected proposal	23.5% (23)
incorrect proposal	66.3% (65)

7.2.2. Case 2: Same as Case 1, Refined Statistical Data

In this test case, we do a similar testing as in the previous one. Again, we use the same self-made error corpus for checking, but we improve the training corpus. We use a refined English version of Wortschatz Universität Leipzig. The changes in the new training corpus are described in chapter 7.1.1. Furthermore, we activate quad- and pentagrams of tokens. For later use we also include the quad- and pentagrams of hybrids. Therefore, we set up a new database and train it with the refined statistical data including the quad- and pentagrams of tokens and hybrids.

Table 7.5 shows the results from the n-gram check of tags, Table 7.6 the hybrid n-gram check results, and finally Table 7.7 the results from the token n-gram check.

Table 7.5.: Test case 2: Error classification of tag n-gram check result

Error class	All sentences	Wrong sentences	Correct sentences
Too small tagset (A)	15.5% (41)	31.3% (41)	0%
Too little statistical data (B)	7.6% (20)	4.6% (6)	10.5% (14)
Erroneous statistical data (C)	0.8% (2)	1.5% (2)	0%
Tagging error during training (D)	0%	0%	0%
Tagging error during checking (E)	1.5% (4)	3.1% (4)	0%
Multiple token sequences with same tags (F)	16.3% (43)	32.8% (43)	0%
Sphere of word too large (G)	4.2% (11)	8.4% (11)	0%
Tokenization error (H)	3.4% (9)	3.1% (4)	3.8% (5)
Style or semantic error (I)	3.4% (9)	6.9% (9)	0%
Correct	57.2% (151)	28.2% (37)	85.7% (114)

Table 7.6.: Test case 2: Error classification of hybrid n-gram check result

Error class	All sentences		Wrong sentences		Correct sentences	
	conventional	upgraded	conventional	upgraded	conventional	upgraded
A	8.0% (21)	0.4% (1)	16.0% (21)	0.8% (1)	0%	0%
B	17.8% (47)	39.4% (104)	8.4% (11)	4.6% (6)	27.1% (36)	73.7% (98)
C	1.1% (3)	0%	2.3% (3)	0%	0%	0%
D	0%	0%	0%	0%	0%	0%
E	1.5% (4)	0.4% (1)	3.1% (4)	0.8% (1)	0%	0%
F	8.0% (21)	0.4% (1)	16.0% (21)	0.8% (1)	0%	0%
G	3.4% (9)	1.9% (5)	6.9% (9)	3.8% (5)	0%	0%
H	3.4% (9)	3.4% (9)	3.1% (4)	3.1% (4)	3.8% (5)	3.8% (5)
I	0.8% (2)	0.4% (1)	1.5% (2)	0.8% (1)	0%	0%
J	2.3% (6)	0%	4.6% (6)	0%	0%	0%
Correct	52.3% (138)	52.3% (138)	35.1% (46)	82.4% (108)	69.2% (92)	22.6% (30)

Table 7.7.: Test case 2 & 3: Error classification of token n-gram check result

Error class	All sentences		Wrong sentences		Correct sentences	
	bi- & trigrams	quad- & pentagrams	bi- & trigrams	quad- & pentagrams	bi- & trigrams	quad- & pentagrams
B	42.4% (112)	47.3% (125)	3.8% (5)	1.5% (5)	80.5% (107)	92.5% (123)
C	0.4% (1)	0.4% (1)	0.8% (1)	0.8% (1)	0%	0%
G	8.3% (22)	1.5% (4)	16.8% (22)	3.1% (4)	0%	0%
H	3.4% (9)	3.4% (9)	3.1% (4)	3.1% (4)	3.8% (5)	3.8% (5)
I	0.8% (2)	0.8% (2)	1.5% (2)	1.5% (2)	0%	0%
Correct	44.7% (118)	46.6% (123)	74.0% (97)	90.1% (118)	15.8% (21)	3.8% (5)

7.2.3. Case 3: Self-made Error Corpus (English), Brown Tagset

This test case uses the revised English training corpus from Wortschatz Universität Leipzig and the TnT tagger, too, but this time with the Brown tagset instead of Penn Treebank. Therefore, we set up a new database once again. Table 7.8 shows the tag n-gram check results, and Table 7.6 the corresponding token check results. The token n-gram check is the same as in test case 2, because the tokens of the training data and the error corpus are the same in both test cases. This means that Table 7.7 shows the tag n-gram check results.

Table 7.8.: Test case 3: Error classification of tag n-gram check result

Error class	All sentences	Wrong sentences	Correct sentences
Too small tagset (A)	1.1% (3)	2.3%3	0%
Too little statistical data (B)	15.5% (41)	2.3% 3	28.6% (38)
Erroneous statistical data (C)	1.5% (4)	1.5% 2	1.5% (2)
Tagging error during training (D)	0.8% (2)	0.8% 1	0.8% (1)
Tagging error during checking (E)	3.8% (10)	5.3% 7	2.3% (3)
Multiple token sequences with same tags (F)	7.6% (20)	15.3% 20	0%
Sphere of word too large (G)	1.9% (5)	3.8% 5	0%
Tokenization error (H)	3.8% (10	3.8% 5	3.8% (5)
Style or semantic error (I)	1.5% (4)	3.1% 4	0%
Correct	62.5% (165)	61.8% 81	63.2% (84)

Table 7.9.: Test case 3: Error classification of hybrid n-gram check result

Error class	All sentences		Wrong sentences		Correct sentences	
	conventional	upgraded	conventional	upgraded	conventional	upgraded
A	0.8% (2)	0%	1.5% (2)	0%	0%	0%
B	23.1% (61)	37.5% (99)	6.1% (8)	1.5% (2)	39.8% (53)	72.9% (97)
C	0.8% (2)	0.8% (2)	1.5% (2)	0%	0%	1.5% (2)
D	0.4% (1)	0.8% (2)	0%	0%	0.8% (1)	1.5% (2)
E	0.4% (1)	1.9% (5)	0%	2.3% (3)	0.8% (1)	1.5% (2)
F	4.5% (12)	1.1% (3)	9.2% (12)	1.5% (2)	0%	0.8% (1)
G	8.7% (23)	1.5% (4)	17.6% (23)	3.1% (4)	0%	0%
H	3.8% (10)	3.4% (9)	3.8% (5)	3.1% (4)	3.8% (5)	3.8% (5)
I	1.1% (3)	1.1% (3)	2.3% (3)	2.3% (3)	0%	0%
J	4.9% (13)	0%	9.9% (13)	0%	0%	0%
Correct	51.1% (135)	51.9% (137)	48.1% (63)	86.3% (113)	54.1% (72)	18.0% (24)

7.2.4. Case 4: Self-made Error Corpus (German)

In test case 4, we train the database with the German version of Wortschatz Universität Leipzig using the Stuttgart-Tübingen tagset. We perform a check with our self-made German error corpus. Table 7.10 shows the results from the tag n-gram check, Table 7.11 the hybrid n-gram check, and Table 7.12 the token n-gram checks.

Table 7.10.: Test case 4: Error classification of tag n-gram check result

Error class	All sentences	Wrong sentences	Correct sentences
Too small tagset (A)	21.9% (57)	43.8% (57)	0%
Too little statistical data (B)	5.0% (13)	0%	10.0% (13)
Erroneous statistical data (C)	1.9% (5)	3.8% (5)	0%
Tagging error during training (D)	0%	0%	0%
Tagging error during checking (E)	2.7% (7)	4.6% (6)	0.8% (1)
Multiple token sequences with same tags (F)	9.2% (24)	18.5% (24)	0%

Table 7.10.: Test case 4: Error classification of tag n-gram check result (continued)

Error class	All sentences	Wrong sentences	Correct sentences
Sphere of word too large (G)	0%	0%	0%
Tokenization error (H)	0.8% (2)	0.8% (1)	0.8% (1)
Style or semantic error (I)	0.4% (1)	0.8% (1)	0%
Correct	57.3% (149)	26.2% (34)	88.5% (115)

Table 7.11.: Test case 4: Error classification of hybrid n-gram check result

Error class	All sentences		Wrong sentences		Correct sentences	
	conventional	upgraded	conventional	upgraded	conventional	upgraded
A	15.8% (41)	3.8% (10)	31.5% (41)	7.7% (10)	0%	0%
B	6.2% (16)	20.0% (52)	3.1% (52)	2.3% (3)	9.2% (12)	37.7% (49)
C	2.3% (6)	2.3% (6)	3.1% (6)	3.1% (4)	1.5% (2)	1.5% (5)
D	0%	0%	0%	0%	0%	0%
E	1.5% (4)	0.8% (2)	2.3% (2)	0%	0.8% (1)	1.5% (2)
F	4.2% (11)	1.9% (5)	8.5% (11)	3.8% (5)	0%	0%
G	5.8% (15)	0.8% (2)	11.5% (15)	1.5% (2)	0%	0%
H	0.8% (2)	0.8% (2)	0.8% (1)	0.8% (1)	0.8% (1)	0.8% (1)
I	0.4% (1)	0%	0.8% (1)	0%	0%	0%
J	2.3% (6)	0%	4.6% (6)	0%	0%	0%
Correct	60.0% (156)	68.8% (179)	32.3% (42)	79.2% (103)	87.7%)114	58.5% (76)

Table 7.12.: Test case 4: Error classification of token n-gram check result

Error class	All sentences		Wrong sentences		Correct sentences	
	bi- & trigrams	quad- & pentagrams	bi- & trigrams	quad- & pentagrams	bi- & trigrams	quad- & pentagrams
B	11.2% (29)	46.2% (120)	3.1% (4)	0%	19.2% (25)	92.3% (120)
C	2.3% (6)	2.3% (6)	4.6% (6)	4.6% (6)	0%	0%
F	1.2% (3)	0.4% (1)	2.3% (3)	0.8% (1)	0%	0%

Table 7.12.: Test case 4: Error classification of token n-gram check result (continued)

Error class	All sentences		Wrong sentences		Correct sentences	
	bi- & trigrams	quad- & pentagrams	bi- & trigrams	quad- & pentagrams	bi- & trigrams	quad- & pentagrams
G	9.6% (25)	1.2% (3)	19.2% (25)	2.3% (3)	0%	0%
H	0.8% (2)	0.8% (2)	0.8% (1)	0.8% (1)	0.8% (1)	0.8% (1)
I	0%	0%	0%	0%	0%	0%
Correct	74.2% (193)	48.5% (126)	68.5% (89)	90.0% (117)	80.0% (104)	6.9% (9)

7.2.5. Case 5: Several Errors in Sentence (English)

In this test case we check sentences which contain more than one error. Results show that it works. We do not give percentage results in a table, because those would be the same as in the previous test cases. LISGrammarChecker points all errors at once if they are recognized by the same type of n-grams. If the errors are recognized by different n-gram valencies, the error which is caused by the smallest n-gram is marked. If one error is corrected and the sentence checked again, the next error is detected and presented to the user.

7.3. Operate Test Cases with Upgraded Program

In this section we perform test cases with an upgraded program. Therefore we use an extended version of LISGrammarChecker. We have implemented more hybrid n-gram checks. The results of test cases 2, 3, and 4 already include these new hybrid n-gram checks. Furthermore we apply rules in addition to the statistical checks. The test results also lead us to specify a new program logic where we combine several program components in a new way. This means that the result of the tag n-gram check triggers new hybrid n-gram checks, and these in turn trigger the rule component. All these extensions are described in more detail in the next chapter, see section 8.4.

7.3.1. Case 6: Self-made Error Corpus (English), Brown Tagset

This test case is similar to test case 3. For a better comparison we use exactly the same statistical training data, the same tagset, and the same error corpus for checking. The only

difference is that we use the upgraded version of LISGrammarChecker. The results are shown in Table 7.13.

Table 7.13.: Test case 6: Results from new program logic

Error class	All sentences	Wrong sentences	Correct sentences
Too little statistical data (B)	22.0% (56)	14.3% (18)	29.7% (38)
Tagging error during checking (E)	2.8% (7)	2.4% (3)	3.1% (4)
Multiple token sequence with same tags (F)	7.1% (18)	14.3% (18)	0%
Sphere of word too large (G)	1.2% (3)	2.4% (3)	0%
Style or semantic error (I)	1.6% (4)	3.2% (4)	0%
Correct	67.7% (172)	69.8% (88)	65.6% (84)

7.3.2. Case 7: Self-made Error Corpus with Simple Sentences (English)

In this test case we train the database with the English version of Wortschatz Universität Leipzig and our self-made composition of texts from ANC, newspapers, and learning English portal. We use the Brown tagset. We check the self-made error corpus with simple sentences. Table 7.14 shows the test results with the new program logic.

Table 7.14.: Test case 6: Results from new program logic

Error class	All sentences	Wrong sentences	Correct sentences
Too less statistical data (B)	4.0% (4)	4.0% (4)	4.0% (4)
Tagger error (E)	3.0% (3)	4.0% (4)	2.0% (2)
Multiple tokens with same tag sequence (F)	14.0% (14)	20.0% (10)	8.0% (8)
Correct	77.0% (77)	68.0% (68)	86.0% (86)

We test the correction proposal in 33 of the incorrect sentences (68%) where the errors are detected. Table 7.15 shows the results.

Table 7.15.: Test case 6: Correction proposal results

Behaviour	Rate
Correct and expected proposal	3.0% (1)
Correct but unexpected proposal	24.0% (8)
Incorrect proposal	73.0% (24)

7.4. Program Execution Speed

We measure the program execution speed. We first take a look at the speed during training. Afterwards we measure the duration in checking mode. All measures are done using only TnT.

7.4.1. Training Mode

The execution time in training mode is represented in a graph which represents the duration of one training block (100 sentences) over the training time and the amount of blocks already stored in the database. The training is done using TnT only. The violet line in Figure 7.1 shows the time to train a block of 100 sentences over the total time while training the Wortschatz Universität Leipzig corpus in English using the Penn Treebank tagset. The graph shows only the half of the training. The overall training time for the corpus is about 17 days. Leaving out the quad- and pentagrams of token, the overall training time is about 10 days. The blue line shows the results for the training of the English corpus from Wortschatz Universität Leipzig, but this time the Brown tagset is used. The training time using the Brown tagset is about 25 days.

The training speed of the German corpus from Wortschatz Universität Leipzig using the STTS is comparable to the English corpus using the Penn Treebank tagset. If more taggers are used for combined tagging, then every block needs about 25 seconds of extra time to execute all taggers and to combine their results.

7.4.2. Checking Mode

Here we measure the time of how long it takes to check a single sentence. This is done for 100 sentences with an average sentence length of about 15 words using the Wortschatz Universität Leipzig corpus in English with the Penn Treebank tagset and the Brown tagset.

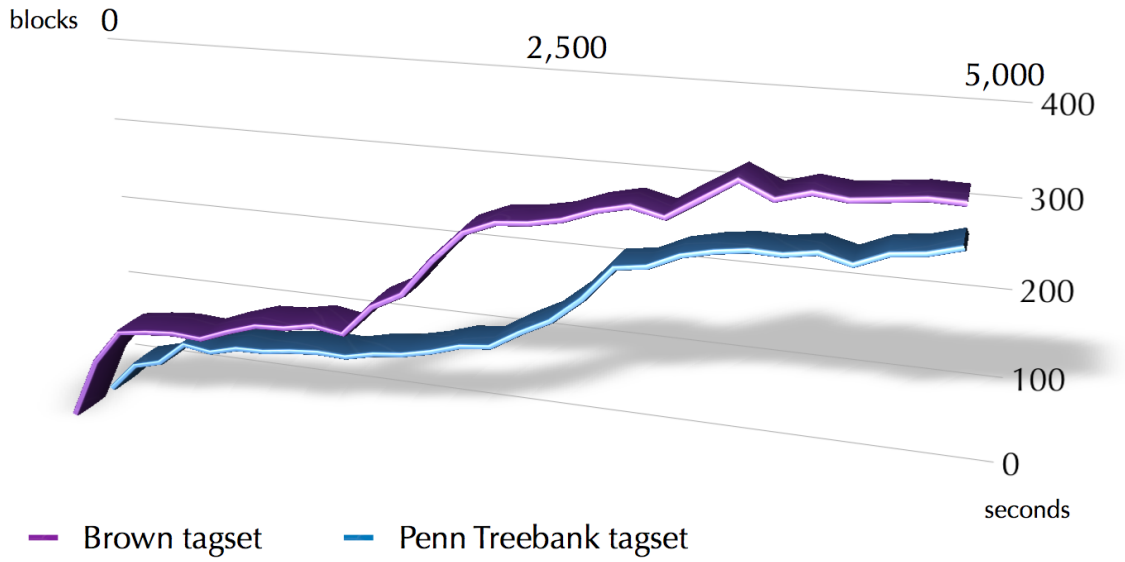

Figure 7.1.: Training time of Wortschatz Universität Leipzig

The same (with also 100 sentences) is done using the Wortschatz Universität Leipzig corpus in German using the STTS. All tests are done using a single tagger—TnT.

Table 7.16.: Grammar checking times

Tagset	1 sentence
Penn Treebank	120 ms
Brown	128 ms
Stuttgart-Tübingen	121 ms

8 Evaluation

In this chapter we evaluate our language independent statistical grammar checking approach. First we review if the requirements for our program are fulfilled. Then we give a detailed explanation about the test results from previous chapter. We show the issues that occurred during the implementation of LISGrammarChecker and in the test cases.

8.1. Program Evaluation

Here, we review our requirements from the analysis section with regard to their fulfillment. Table 8.1 overviews our developed requirements with their consequences and evaluates if each requirement is fulfilled ✓ or not ✗. Symbol ❍ marks entries, where we cannot absolutely say that it is fulfilled, but we can also not say that it is not fulfilled. These entries are ambivalent and we explain them in more detail.

Table 8.1.: Fulfillment of established requirements

Requirements	Consequences	Fulfilled
Language independence	Process statistical data separately for every language	✓
	Save huge data permanently (database system)	✓
Gain data and save it permanently	Separate training mode to save data	✓
Grammar checking without saving wrong input	Separate grammar checking mode	✓
Correct statistical data	Gain correct text from reliable sources	✗
Program execution speed	Fast program components (programming language)	✓
	Short data access time (fast data storage)	❍
Accurate reliable data	Much statistical data	❍
Use Internet n-grams	Possibility for Internet queries	✓
Tagged input text with high accuracy	Integrated tagger	✓
	Combined tagger	✓
User need to interact with the system	Appropriate user interface to give input and preferences	✓
Show results (training successful or errors found)	Output result to the user	✓
Save data in training mode for later grammar checking	Algorithm to extract information from input	✓
Perform grammar checking in grammar checking mode	Algorithm to perform grammar checking	✓
Few false positives	Use thresholds for error calculation	✓
Propose a correction	Gain proposals from the most likely alternatives out of the statistical data	✓

8.1.1. Correct Statistical Data

LISGrammarChecker works only as precisely as the statistical data in the database. In order to allow an accurate grammar check we need good quality data for training. That means that

the texts should be grammatically correct and of good language style. Thus, it is not possible to use every available corpus resource.

We developed the requirement for LISGrammarChecker to use only correct statistical data to build up the database. We intended to fulfill this requirement by looking for adequate corpora. We needed a lot of time to get appropriate texts for training. Many corpora are not freely available due to copyright issues.

The following reasons show that we need to mark the requirement as not fulfilled. All sources that are a possibility, i.e. are large enough and freely available, and thus could be used to build the database claimed that their corpora are checked for mistakes. Therefore we assumed that they are correct. But all corpora we use contain severe mistakes. The corpus from Wortschatz Universität Leipzig for example contains headlines of newspapers and lines which represent stock data like "DAX 4,302 +5.2% up BWM 50,02 +3.2% up". These text lines are counterproductive for our approach because our program works just with full and intact sentences. Another problem in these corpora constitute typographic mistakes. The misuse of French accent characters as quotation marks or apostrophes cause problems, e.g. clitics cannot be split, tagging errors occur, and tokens do not match.

8.1.2. Large Amount of Statistical Data

The requirement of accurate and reliable data means that we need to train enough statistical data to get accurate grammar checking results. This requirement is not fully fulfilled. It was possible to train a huge amount of data. The handling using the database system works very well and the query time in checking mode is still low even if there are the data of 1 million processed sentences stored. The tests show that 1 million sentences are still not enough to get overwhelming results and there is a need for more training data.

The training of more data is possible but it will be very time consuming on a standard computer. During the training of one million sentences more than 4 billion database transactions are done. The needed time for each database transaction gets longer if there are more data in the database table. Therefore, the requirement could not be fulfilled better because of the lack of time to train.

8.1.3. Program Execution Speed

Another requirement which is not thoroughly fulfilled is the program execution speed. The time measurements in the last chapter (section 7.4) show that it can take up to 20 days to train a corpus of about 20 million words.

The measures are done using only TnT. If the combination of taggers would be used, the time would be 10 times higher because Stanford Tagger and the combination needs some time. Using the combination of the grammar checker in checking mode does not fulfill the requirement of fast program execution.

Thus it is possible to check sentences in a time which is short enough to support the check in realtime. This fulfills our requirement. The training of the corpora takes a long time, i.e. the demanded execution time is not reached in training mode.

8.1.4. Language Independence

In general, LISGrammarChecker is language independent because the algorithm works for all tokens regardless of the language or alphabet used. Furthermore, the statistical input texts can be given into the grammar checker in every language. But even if this is working, some processing steps are more or less language dependent.

The most problems are caused by the tokenization step. The determination of the token and the sentence boundaries is language dependent. Furthermore, not all languages use the same punctuation. The used tokenization script is specialized to European languages.

Another not fully language independent part is the taggers. TnT, Treetagger and Stanford Tagger are basically language independent. Nevertheless they need to be trained for every language before they can be used for the specific language. It could be also a possibility to add another tagger for the wanted language. But if the use of a tagger is not possible at all, the approach can be used just with the token n-grams functionality including the results from an Internet search machine. Thus, only the tokenization problems need to be solved in order to use a new language with this approach.

8.1.5. Internet Functionality

The requirement to use Internet n-grams is fulfilled. We have realized the possibility for Internet queries which works precisely as it should. But under some circumstances it does not work as expected. This could be the case if this functionality sends too many requests to a search engine. This means for example that too many Google requests could result in a temporary ban to do more requests. Google can be asked if they allow the use of their API for this purpose to solve this issue. We counteract this problem by saving already sent requests to our database so that a request does not need to be sent twice. This storage functionality solves also the issue that Internet requests are slower than local database queries.

Internet n-grams can differ in various aspects such as quality, amount or reliability. The main reason are different sources and different users. For example Google is used by everyone and

thus everyone's writings are basis for search results. Google Scholar in contrast is used by a smaller group of people. The sources are e.g. research paper which are more well-founded. This means that the results from Google Scholar might be more correct, but there will be fewer results. Quantity and accuracy are important for reliable data. Without further tests it is thus not definable with reasonable certainty which search engine gives statistically more reliable results.

8.1.6. Encoding

A disadvantage of LISGrammarChecker is the lack of special encoding functionalities. In general the program is compatible with all available encodings on a certain system but at the moment there is no possibility implemented which distinguishes between different encodings. This means that the first input text to train the grammar checker specifies the encoding. If the grammar checker is trained with another text in a different encoding, there could be a mismatch which leads to an incorrect tagging and therefore causes problems with the accuracy of the grammar checking. This is an issue even if English text is used. This is mainly due to to use of typographic quotation marks. A solution for this problem is the re-encoding of the texts with Unicode (UTF-8).

8.1.7. Tokenization

With the used tokenization script there exist some issues as follows:

Lack of Memory The script reads the whole file at once and thus needs a lot of memory. If large texts are passed to it, it is quit by the operating system due to lack of memory. The script needs to be rewritten to cache parts of the data on disk.

Detection of Sentence Boundaries The detection of sentence boundaries is still not perfect. Cardinal numbers at the end of a sentence are often interpreted as ordinal numbers and therefore the end of the sentence is not detected. In our case the training corpus contained one sentence per line. Substituting the period with an exclamation mark improved the detection of the boundaries greatly.

Quotation Quotation is not always separated and treated as an individual token. If a single quotation mark is situated on the left side of a token, it is not separated.

Different tokenizers in the taggers In our implementation Stanford Tagger uses a different tokenization method which is provided with the tagger. This causes some problems with the combination of all taggers. The combination is only successful if the whole input corpus is tokenized the same way so that all tokens match.

Multiwords Multiwords are never treated as one token.

8.2. Error Classes

We specify several error classes to classify the errors that occur in the test cases. They describe the reasons why LISGrammarChecker does not give the best result in a specific case. Not all error classes make sense for every n-gram check, i.e. token, tag and hybrid n-gram check. We also try to give solutions how to avoid a specific error type.

Too small tagset (A) This error class means that the tagset does not represent all morphological features of the used language. The categories, i.e. the tags, that classify the tokens are too inaccurate. This means that some morphological features get lost when a token is assigned to a certain category. This is usually an issue when checking the agreement between two words. In the Penn Treebank tagset for example the distinction between singular and plural is not possible in all word classes. One example for that is the word class of determiners (DT). The words these, this, the, and a are all tagged with the tag DT. No syntactical features (like indefinite or definite article or plural or singular) are encoded in the tag. This means that although determiner and noun in the trigram "these (DT) money (NN) problem (NN)" do not agree, it is not possible to detect the error by just using the tags.

The easiest way to avoid such problems would be a tagset containing more tags and which encodes more morphological features. In English, the Brown tagset could be used. In German the Münsteraner tagset is large enough.

Too little statistical data (B) If an error occurs because of a too small corpus which is used to train the grammar checker, i.e. does not contain all necessary data, we talk about too little statistical data. Due to this lack of data, some n-grams cannot be found in the database. Especially old and rare words lead to errors during the checking process because of unknown words. For example, if the training corpus does not contain a certain word, all n-grams that include this word are not found. Another issue is the use of short forms for some words: 'll for will or 's for several words. When using statistics, these are distinct tokens. Thus, if the word will is trained, its abbreviation is still not found while checking and therefore considered as wrong.

These problems—unknown word and short forms—can be solved through training a correct sentence which includes the missing token. Many errors of this type only occur because of the source of the statistical data. The corpora we use are mostly from newspapers. It is thus quite usual that sentences from other fields are not found properly.

Erroneous statistical data (C) In this error class we collect the errors which are caused by incorrect statistical data. If the training data contain one mistake, 41 wrong n-grams are written to the database. Some n-grams which are compared against the database could be matched with the erroneous n-grams and thus are categorized as correct. For example if the sentence "He is is a boy." is part of the training corpus, LISGrammar-Checker accepts the bigram "is is" as well as the tag bigram "VBZ VBZ" which leads certainly to wrong assumptions.

One solution is to correct the corpora by hand. As an alternative, it seems to be quite impossible to solve this problem by using statistics, because e.g. the bigram "is is" is found more than 15 times in the corpus Wortschatz Universität Leipzig. Compared to some rare constructions this can be denoted as often.

Tagging error during training (D) This error class describes cases, where the tagger assigns a wrong tag to a certain word during training. This can happen for various tokens which are ambiguous in their part-of-speech. An example is the word show which can be a noun NN or a verb VB. If the wrong tagging occurs in the training phase, there are a couple of wrong n-grams of tags written to the database.

Tagging errors could be minimized by using a single tagger that yields a higher accuracy or an appropriate tagger combination. While there is no tagger or tagger combination which can achieve an accuracy of 100%, this issue remains and is not completely avoidable.

Tagging error during checking (E) This error class is similar to the previous one. The main difference is the part of the program where the error occurs. Here the wrong tagging takes place in checking mode. The sentence which is going to be checked contains one or more tokens which are labelled with the wrong tags. To solve this error type, the same considerations as for error class D apply.

Multiple token sequences with same tags (F) In this class we classify all erroneously as correct accepted n-grams which are caused by sequences of tags which are valid but the corresponding token sequence is not valid. The example sentence "He has been qualified to president of the country." is not correct and is tagged with the following tags in the Penn Treebank tagset: "PRP VBZ VBN VBN TO NN IN DT NN .". The correct sentence "He has been put to death by a cowboy." is tagged with exactly the same tag sequence but this sentence is correct. Thus the error is not detected. This error class is very similar to error class A—too small tagset. Strictly speaking, A is a subset of F. The main difference is that A occurs because of too less morphological features in the tagset and thus can be solved by a larger tagset.

Due to the variety of combinations in natural languages, error F can only be minimized but not completely solved.

Sphere of word too large (G) This error says that the influence of a word is larger than the n-gram which is used to check. For example subject and verb in sentence "Jim and the young cute girl plays a game." do not agree. If we check a pentagram as the largest n-gram we cannot find the agreement between the subject and the verb. The first pentagram which covers the verb is "the young cute girl plays" and this pentagram is correct.

To solve this type of error larger n-gram windows could be used.

Tokenization error (H) A tokenization error occurs e.g. if the sentence boundaries are not found. If the end of the sentence is not found, the program runs accidently into the next sentence. Furthermore, if it is the last sentence to check and the end is not found, the sentence is not checked at all.

A better tokenization method can minimize the amount of tokenization errors. But like the tagging problem, there exists no tokenizer which is capable to ensure a tokenization with 100% accuracy in English or German.

Style or semantic error (I) This error class is similar to class F. Instead of a wrong grammar, the semantic of the sentence is not correct, like in the following example: "Green ideas sleep furiously.". These types of semantical errors cannot be detected by n-grams of tags. Nevertheless, LISGrammarChecker detects some semantic errors during token n-gram check, even if their detection is not our main goal.

Check not useful (J) A not useful check occurs only for hybrid n-gram checks. If there is already an n-gram of tags which is not found in the database, there is no chance to find the corresponding hybrid n-gram. If a tag n-gram is not found, the hybrid n-gram check is skipped.

We use these error classes to interpret and evaluate the test case results in the next section.

8.3. Evaluation of Test Cases 1-5

Our test case results are multifaceted. In this section we interpret all results from section 7.2. Therefore we show the general working of LISGrammarChecker. We interpret the first three test cases which are real world examples for English. They differ in the training corpora and the tagset. To interpret the results when LISGrammarChecker does not give the correct grammar checking result, we make use of the error classes from the previous section. Some of these classes do not make sense in every check. Furthermore we present the results of a check in German (test case 4). We show how LISGrammarChecker handles more than one error in a sentence in test case 5. Finally our results lead to an upgrade of LISGrammarChecker, where we add rules and more hybrid n-grams.

Two example sentences We start with two example sentences of test case 1—a correct and an incorrect one. Listing 8.1 presents an excerpt of the output of LISGrammarChecker when the incorrect sentence "These also find favor with girls, whose believe that Joan died from a heart attack." is checked. The grammar checker does not find the tag bigram "WP$ VBP" and thus the corresponding phrase is pointed out as an error.

```
1 Analyze tag n-grams in sentence "These (DT) also (RB) find (VB) favor (JJ) with (IN)
      girls (NNS) , (,) whose (WP$) believe (VBP) that (IN) Joan (NNP) died (VBD) from
       (IN) a (DT) heart (NN) attack (NN) . (SENT)":
2
3 1 unknown 2-consecutive-tags combination(s) found:
4 whose believe
5
6 Overall error counter for tag n-gram check is 12.
```

Listing 8.1: Check incorrect sentences

In Listing 8.2 we show an excerpt of the output of a correct sentence check for the sentence "These also find favor with girls, who believe that Joan died from a heart attack.". All n-grams are found in the database and thus no error is pointed out. These two examples show that the approach itself works.

```
1 Analyze tag n-grams in sentence "These (DT) also (RB) find (VB) favor (JJ) with (IN)
      girls (NNS) , (,) who (WP) believe (VBP) that (IN) Joan (NNP) died (VBD) from (
      IN) a (DT) heart (NN) attack (NN) . (SENT)":
2
3 Overall error counter for tag n-gram check is 0.
```

Listing 8.2: Check correct sentences

Overall error threshold The overall errors of both sentences show optimal values. The incorrect sentence has a high value, the correct one a low. The overall error depends on the individual n-gram error weights. In our tests we tried to find optimal individual weights and an optimal overall threshold. To perform these tests we used the evaluation script described in section 7.1.3. We figured out that the threshold in its current implementation is not always meaningful even if the individual weights are carefully selected. The reason for not being meaningful is the lack of adaptation to e.g. the sentence length or multiple errors in a sentence. For example if a sentence has several incorrect token pentagrams at different positions, this is not necessarily an error which should be marked because these errors could be caused by too little statistical data. Thus we want the token pentagram error weight low. But as several errors are only detected by incorrect token pentagrams, the solution to set the token pentagram error weight low does not solve the problem. We have a problem to find individual error weights that fit all error types. Even adapting the error weight to the sentence length, e.g. with a local error threshold, the results are not sufficient enough. Thus we do not regard the error thresholds in our test cases but regard all individual n-gram errors instead.

n-gram checks (test case 1) In our first test case we do not include the token quad- and pentagrams because we thought that these need too long for training but are not effective enough for that effort. We take a look at the general effectiveness of each n-gram check. The test results reveal that the tag n-gram check (here we use the Penn Treebank tagset) detects only about a fourth of the errors. The hybrid n-grams are only little better with correctly detected errors about a third of the time. The best detection rate with about 75% is the token n-gram check (here we use only tri- and bigrams). This rate is not bad, but we need to take a look at the side effect, i.e. the correct sentences which are erroneously declared as incorrect—the false positives. In more than 80% of the correct sentences is at least one incorrect token n-gram. The hybrid n-gram check classifies only around 25% as incorrect. The best result is achieved by the tag n-gram check where only few false positives occur.

Primary errors We learn that most errors are caused by too little statistical data. This problem exists for the tag n-gram check but it is even more severe for the token n-gram check. The reason for this is that there are not as many possibilities for tag sequences, e.g. the Penn Treebank tagset has 48 tags which can be used to build sequences but there are many more possibilities when using tokens. This means that the tag n-gram method can be trained with less data to cover all possibilities and thus the amount of statistical training data is not such a severe problem as for the token n-gram check. There is only a problem when using the tags of a whole sentence. The sentences can be built up in so many ways that the possible tag sequences are numerous. The test results show that the sentence tag sequence of about 80% of the correct sentences are not found. We regard the sentence tags separately and do not include them in the tag n-gram check result.

Two other issues appear for the tag and hybrid n-gram check: too small tagset and multiple token sequences with same tags. These problems cause many incorrect sentences to not be considered as incorrect. Using the Penn Treebank tagset this happens often because a lot of morphological features are not encoded in the tags of the tagset. For example the correct sentence "Those are worse." is tagged with the tag sequence "DT VBP JJR". The same tag sequence is used for the sentence "The are worse." which is incorrect. If the first correct sentence is trained, the second and incorrect one is classified as correct during checking. The error of a multiple token sequence with the same tags is very similar to a too small tagset error. The unique feature of a too small tagset is that the tagset does not represent all morphological features of the used language.

We see that the amount of detected errors which are caused due to a problem with the sphere of a word is fewer if we use tag pentagrams. At the beginning the cost and effort to use larger token n-grams seemed to be too high compared to the amount of the sphere of word errors. But we learn that a test with quad- and pentagrams of

tokens would be interesting. Thus we specify a second test case where we take a look at the token quad- and pentagrams (see below).

Secondary errors The denotion secondary does not mean that these errors are not important. We want to express that these errors are less relevant for our conclusion as they are not primarily caused by our approach but by external sources. For example, erroneous statistical data are annoying, but our approach depends on huge statistical data and thus this trade-off needs to be accepted until a better training corpus is available. This is similar regarding tokenization errors. We try to avoid this type of error in the next test case through refining the training data, but as this is manual work, it is very time-consuming and thus only possible up to a certain extent. A last error type of this family are tagging errors caused by the taggers. These can be lowered by combined tagging, but a tagging accuracy of 100% is not possible. Unfortunately, we need to accept these secondary errors.

Correction proposal The current implementation of the correction proposal supports the search of an alternative by a wildcard in the middle of the not found n-gram. This does not support a meaningful correction proposal for errors where an additional word is inserted (like "He is is very young.") or a word is skipped (e.g. "She to go."). Due to this restriction nearly 50% of the wrong sentences could never get a useful correction proposal. About 15% of the remaining incorrect sentences get a correct proposal in the first test case. About one third get a proposal which is at least grammatically correct and fits. For the remaining errors an alternative is proposed which is not useful at all. Thus the idea to propose corrections works only roughly and needs to be refined.

Refined training data (test case 2) For test case 2 we trained the database with refined statistical training data to avoid problems like erroneous statistical data and tagging errors during training. It is not possible to avoid the last issue completely because the tagger has only an accuracy rate of about 96.7% and therefore causes errors even if the training data would be 100% correct. The results show that the refined statistical data lower the amount of erroneous statistical data and wrong tagged text.

The token quad- and pentagrams detect more errors but at the same time there are more false positives. Table 7.7 shows that the token n-gram check is not sufficient with the current statistical training data. To lower the errors of too few statistical data, we need much more training data (e.g. the Google n-grams [BF06]).

The refinement of data does not change many of the results for the tag n-gram check. Therefore, the Penn Treebank tagset remains insufficient and the errors of a too small tagset remain. Thus we use the same statistical training corpus but use a larger tagset—the Brown tagset—in the next test case.

Larger tagset (test case 3) To verify our hypothesis that a larger tagset gives better results, we use the Brown tagset in test case 3. The tagset provides more tags which represents more morphological features.

As we can see in table 7.8 the detection rate of the tag n-gram check rises to about 60% with this larger tagset. Most errors of a too small tagset disappeared. The remaining errors are e.g. due to the indefinite determiner a and an where the Brown tagset does not make a difference. With the larger tagset, the tagger has a lower accuracy rate and thus, as we can see in the table, more tagging errors occur.

Unfortunately the false positive rate increases. The tag n-grams containing the sentence start and end marker are the main reason for that. Furthermore the problem of too few statistical training data causes false positives. A training corpus containing all grammatical structures could lower the false positive rate significantly. The tags of a large tagset represent more morphological features which leads to more accurate tag sequences and more possible tag sequences. This is an advantage—more errors are detected—and a disadvantage—more data are needed to cover all possibilities—at once.

The hybrid n-gram check shows the same. The detection of mistakes is better but the false positives rise.

Adverb-verb-agreement The adverb-verb-agreement uses the tag which marks an adverb to determine temporal adverbs. The problem is that this specification is not sufficient—more than the temporal adverbs are determined. This problem exists in the Penn Treebank tagset because there all adverbs are classified with the same tag. In the Brown tagset there is an individual tag for temporal adverbs but this does not include all key words that trigger specific verb forms. This functionality could be refined by using key words instead of the temporal adverb tags to determine the appropriate temporal adverbs that trigger specific verb forms.

German test (test case 4) The German test shows that our approach can be used with different languages. It works similar to the English one and the results are comparable.

Like the Penn Treebank tagset, the STTS does not contain enough tags to represent all morphological features. In German this is even worse than in English because the German language is morphologically more complex and thus there can be more mistakes due to disagreements. Table 7.10 verifies this. About 60% are incorrect because of a too small tagset or a token sequence which has the same tags.

Adjective-noun-agreement The adjective-noun-agreement does not make sense for English because this language does not distinguish adjectives with respect to number, gender, or case. But this functionality can be used in German where enough morphological

features are available. Test sentences show that the idea works. If we test more sentences the problem in most cases is too few statistical data. Thus it would be better if the tags are used instead of the tokens. Therefore the tagset need to support these features—STTS does not. The Münsteraner tagset would solve this problem because it provides the necessary features. Unfortunately, we cannot perform tests with this tagset because we have no data available to train a tagger with it.

More errors in a sentence (test case 5) We take a look at the capability of LISGrammarChecker to handle more than one error in a sentence. While using LISGrammarChecker we have already seen more than one indicated erroneous phrase. Now we analyze this functionality in detail. Results show that it works—all errors in a sentence are detected. But the accuracy depends on the same issues as if there is only one error in a sentence. LISGrammarChecker indicates all errors at once if they are recognized by the same type of n-grams. If the errors are recognized by n-grams with different valencies, only the error which is caused by the smallest n-gram is indicated. If this error is corrected and the sentence checked again, the remaining errors are still detected and the next error is displayed to the user.

First results lead to a program upgrade We have shown that the grammar checker works for different languages. At this point we know that the error threshold in its current implementation is not meaningful. The training data which is available for us does not allow the use of a token n-gram check with the required accuracy. Furthermore, the use of a tagset which contains tags for all available morphological features of the used language is recommended.

All those results until now lead us to extend LISGrammarChecker in several ways. On the one hand we propose to add more hybrid n-grams to the statistical approach. On the other hand the combination of our approach with rules would be interesting. Finally we propose to combine several program components in a new way, i.e. the results of the tag n-gram check trigger the check of hybrid n-grams and those influence the application of rules. In our opinion these upgrades could give a larger revenue. Below, we test these ideas.

8.4. Program Extensions

Our evaluation results lead us to implement further functionality to LISGrammarChecker. We have upgraded LISGrammarChecker with additional hybrid n-gram checks. Now we regard the hybrid pentagram with a token in the middle and the two tags of the two left neighbored tokens and the two tokens of the two right neighbored tokens. The second additional hybrid n-gram is a quadgram made of the sequence of a tag as first, then two tokens

and a tag again. Furthermore we have included two types of rules—rules to verify errors and rules to verify the correctness of a sentence. Finally we have combined all functionalities to a new program logic. In subsection 8.4.4 we describe how we combine the tag and hybrid n-gram check with both types of rules.

We have many more ideas of what could be added to our program. We have extended the hybrid n-gram check and added rules, and now LISGrammarChecker can be used with more than one database. We think that these extensions give the highest revenue, i.e. the improvement of our program is noticeable. Further extension ideas that we do not implement are described in the subsequent chapter about future work.

8.4.1. Possibility to Use More Databases at Once

Because of our results of the execution speed, we see that training needs long time, we have used more computers to train the statistical database. These databases are very large and a merge would cause a time delay. Thus we considered the possibility to use LISGrammarChecker with more than one database.

We implemented this by extending the module data.database. The extended function establishDatabaseConnection now establishes not only one database connection, but more. We realized this by a second handle, as the listing shows.

```
1 // Initialize MySQL
2 mysql = mysql_init(null);
3 mysql2 = mysql_init(null);
4 ...
5 // Connect to MySQL database
6 // Parameters: handle, host, username, password, db name, port, unix socket, clientflag
7 mysql_real_connect(mysql, "localhost", "lis_grammar_checker", "sgchecker", dbname.ptr,
      3310, null, 0)
8 mysql_real_connect(mysql2, "localhost", "lis_grammar_checker", "sgchecker", dbname2.ptr,
      3310, null, 0)
```

Furthermore we have extended all functions that read data (all `getEntryFromDBNAME` functions) insofar that these read from both databases and combine the results.

8.4.2. More Hybrid n-grams

LISGrammarChecker uses three hybrid n-grams: A trigram which consists of a token with its left and right neighbors tags (tag-token-tag), a bigram of a token with its left neighbors tag (tag-token), and a bigram of a token with its right neighbors tag (token-tag). Our idea is an extension of the hybrid n-gram check. We introduce two more hybrid n-grams: A

pentagram which consists of a token with its two left and two right neighbors tags (tag-tag-token-tag-tag) and a quadgram which consists of two neighbored tokens with a tag on each side (tag-token-token-tag).

To use these two additional hybrid n-grams, both the training and checking mode of LISGrammarChecker needs to be extended. In training mode these two new n-grams are extracted and stored to the database. In checking mode this stored information is used to make further checks similar to the existing ones. The databases need to be retrained in order to make use of the new hybrid n-grams.

8.4.3. Integration of Rules

We integrated a set of rules into LISGrammarChecker. These are of two types—some rules verify the correctness of a sentence and some verify that there is an error in a sentence. All rules are applied after the statistical approach has done its task. The phrases which are declared as erroneous by the statistical part serve as input to this rule component. They include both tokens and tags, separated by a pound sign #. The format is shown in the following example:

```
1 Peter#NNP#has#VB#a#DET#ball#NN#.#SENT
```

Listing 8.3: Input to the rules component

The rule component consists mainly of the two functions `verifyErrorWithRules` and `verifyCorrectnessWithRules`.

In `verifyErrorWithRules` we use regular expressions to verify the detected errors which serve as input. The regular expressions that represent the rules are stored in a simple text file rules_to_verify_errors.text. The regular expressions can access both the tokens and the tags and use them to apply rules. We provide several rules. Listing 8.4 shows three example rules to check if the word after the determiners a and an starts with a vowel and if there is a missing verb after the word that.

```
1 ^(([^#]+#[A-Z$0-9]+#)*)an#[A-Z$0-9]*#[^aeiou][^#]+#[A-Z$0-9]+ // Rule 1
2 indefinite article "an" needs a vowel at the beginning of the following word (use article "
      a" instead). // Notice 1
3 ^(([^#]+#[A-Z$0-9]+#)*)a#[A-Z$0-9]*#[aeiou][^#]*#[A-Z$0-9]+ // Rule 1
4 indefinite article "a" cannot be followed by a word which starts with a vowel (use article
      "an" instead). // Notice 2
5 ^(([^#]+#[A-Z$0-9]+#)*)that#CST#(([^#]+#(N[A-Z$0-9]*|CC))*)#[^#]+#[^V] // Rule 3
6 a verb after "that" is missing // Notice 3
```

Listing 8.4: File rules_to_verify_errors.text with regular expressions

The second type of rules that we have implemented, verifies the correctness of a sentence phrase. Function `verifyCorrectnessWithRules` gets the as erroneous marked phrases as

input which consists of the tokens and the corresponding tags. Rules that we have already implemented check e.g. if a proper noun and a verb agree, or if the verb is in third person singular after a singular noun. The regular expressions are also stored in a text file, in rules_to_verify_correctness.text. Example rules to verify the correctness of a sentence are as follows:

```
1 ^([^#]*##[A-Z$0-9]*##)*[^#]*##NNP##[^#]*##VB(P|Z)?(##[^#]*##[A-Z$0-9]*)* // Rule 1
2 Agreement between proper noun and verb // Notice 1
3 ^([^#]*##[^(CC)]##)*[^#]*##NNP##[^#]*##VBZ(##[^#]*##[A-Z$0-9]*)* // Rule 2
4 3rd person singular after singular noun // Notice 2
```

Listing 8.5: File rules_to_verify_correctness.text with regular expressions

The two files contain the rules as well as descriptions. The odd lines contain the rules and the even lines the corresponding descriptions. If a rule is applied, the description is printed as a notice to the user. This approach is easy extensible by inserting new rules with corresponding notices into the text files.

8.4.4. New Program Logic: Combination of Statistics with Rules

At the beginning we thought that all three n-gram checks would reveal overlapping errors but also distinct ones and thus we regard all those checks separately. We have learned from our tests that a combination of the individual n-gram checks could give more accurate results. As we do not have enough statistical training data that the token n-gram check suffices we focus on the tag and hybrid n-grams. The main problem if the token n-gram check has too little statistical data is the false positives rate. Thus our new program logic combines the tag n-gram check with the new hybrid n-gram check in a new way. Furthermore additional rules are integrated into the new workflow. The new program workflow is shown in Figure 8.1.

In this combination the tag n-gram check works as before. If the n-gram of tag check detects a sentence as wrong, the erroneous part of the sentence is printed out. But if it classifies the sentence as correct it is passed to the newly introduced hybrid n-gram check using a token with two tags on each side or to the second new hybrid n-gram check. Depending on the results, the sentence is marked as correct or it is passed to the rules check. In this last step rules are applied. Depending on the type of rule, the sentence is treated as correct or wrong. If no rule is applied the sentence is handled as correct.

8.5. Evaluation of Upgraded Program

In this section we interpret the test case results of our upgraded program. We analyze the new hybrid n-gram check as well as the rules. A comparison between the previous version

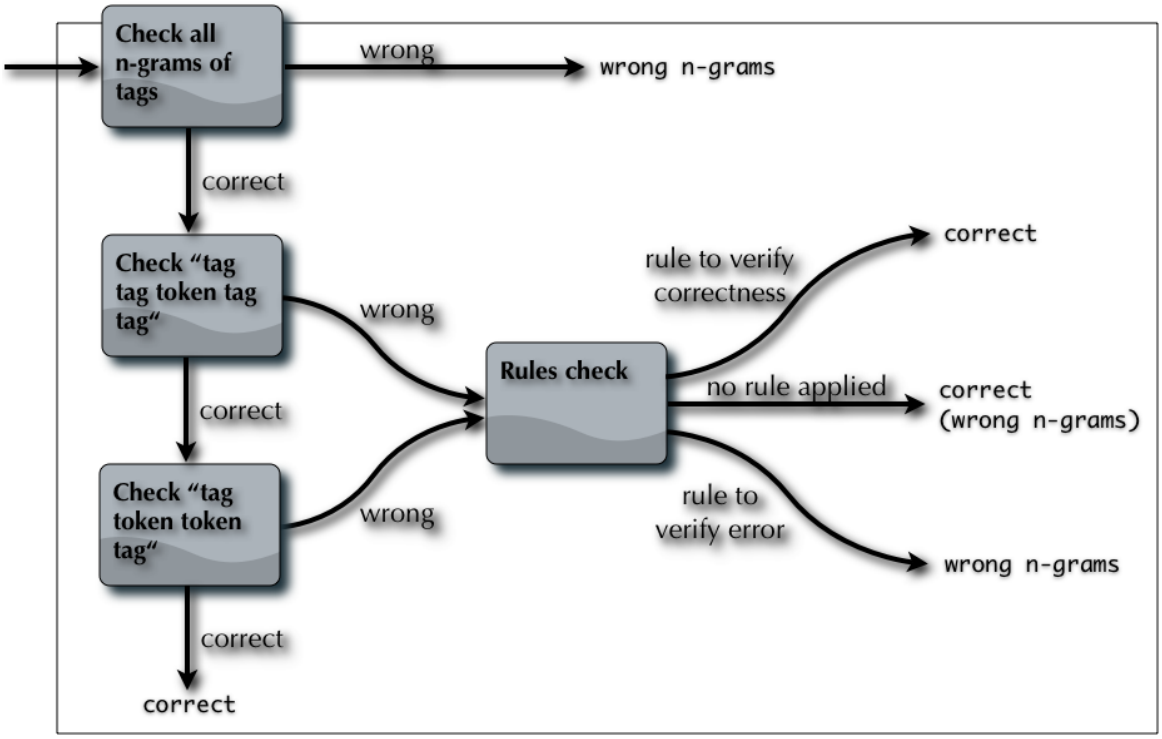

Figure 8.1.: New program logic

of LISGrammarChecker and the new logic implementation shows the improvements.

New hybrid n-grams (test cases 2 and 3) The result tables of test cases 2 and 3 already include the new hybrid n-gram check results. These show that the new hybrid n-grams find more errors. Using the Penn Treebank tagset, more than twice the errors are detected compared to the old hybrid n-gram check using just bi- and trigrams. This means that more than 80% of the errors are found. Using the Brown tagset, the detected errors are also far above 80%. Unfortunately the false positive rate raised to about 80% which was about 30% (Penn Treebank) and 45% (Brown tagset) when using bi- and trigrams. This is caused by the lack of statistical data. As the new hybrid n-gram check is not constructed to be used separately, we do not overstate these results.

Rules (test case 6) Rules are always language dependent and if the rules are used with tags, they even depend on a specific tagset. This is a restriction for LISGrammarChecker on the one hand, but on the other hand the rules make an impact and improve the results. The results show that the amount of sentences correctly classified as wrong is higher. That means that the effectivity of the program is higher because of the program extensions. The already included rules to verify if a sentence is wrong are capable of

detecting 8 sentences which are usually not detected by the statistical approach itself. Therefore, the rate of the detected sentences increases by about 10 percentage points.

New program logic The new program logic, i.e. the combination of the methods works properly and the results from test case 6 show an improvement. We assume from the previous results that it is possible to reach an even higher detection rate if more rules are added to the rules component. One unresolved issue, even in this implementation, is that there is no improvement concerning the false positive rate. To avoid such errors one needs to consider implementing a postprocessing rule functionality for the sentences which are declared as wrong by the tag n-gram check unless no larger training database is used.

Simple sentences (test case 7) The check of the simple sentences error corpus shows very positive results. Table 7.14 reveals that many—almost 70%—of the erroneous sentences are detected correctly. The false positives are still not completely gone, but the rate shrank to less than 15%. Reasons for the false positives are mainly due to multiple tokens with the same tag sequence, but also due to tagging errors and too little statistical data.

If we regard the errors in detail, we see that there are two main problems. The first concerns the tagger. Newspaper texts were used to train the tagger. If we are using simple sentences which are completely different to newspaper text, the tagger is less accurate and thus causes tagger errors and increases the multiple tokens with same tag sequence errors. The second problem concerns the tagset. The tagset does not completely conform to our requirements. There are e.g. many different classifications for nouns. The Brown tagset supports the distinction of normal, locative, proper, temporal, and some additional types of nouns. For our approach this is not needed and is even counterproductive. The possible tag sequence combinations are too many to be covered with the available statistical data. This means that the sentences "The company is small." and "The house is small." are tagged differently. Not all of these nouns are necessary to suffice the n-gram checks and the need to train the same sentence structure with every type of noun leads to false positives. For us this means that we need to think about a tagset that conforms completely to our requirements, i.e. supports all needed features for the checking but not more.

Correction proposal The correction proposal results from section 8.3 are confirmed. There are problems with too few statistical data and the approach is not always useful. In the next chapter we will discuss some new ideas how this feature can be refined in a way to give better results.

Concluding results demand a combination of statistics with rules Every n-gram check has advantages and disadvantages. The high false positive rates of the hybrid and token n-gram checks make it necessary to find a method to distinguish between real mistakes

and false positives. We have found a solution in using rules. The idea to trust the tag n-gram check with only few false positives and the use of the hybrid n-grams in an approved way works very well. It is clear that more training data and more rules improve the overall performance. The more statistical data are available, the less rules are needed. But even if there are an unlimited amount of trained data available, the use of rules is still necessary in order to detect all errors. Another important step is the usage of an appropriate tagset. This tagset should be customized to the actual usage to avoid unnecessary false positives.

Part IV.

Concluding Remarks

9 Conclusion

We know that grammar checking is important in various aspects. Common grammar checkers are rule-based. We consider the new field of statistical grammar checking. Therefore we hypothesize that it would be possible to check grammar up to a certain extent by only using statistical data, and this independent from a natural language. This hypothesis has been proved in large part.

We have shown that LISGrammarChecker—our Language Independent Statistical Grammar Checker—works for different languages and is capable to handle even multiple errors of several types in one sentence. English and German are checked as proof of concept, but other languages can be used similarly. It is very important to train enough and accurate statistical data. But it is difficult to gain enough freely available data which are in sufficient high quality. The possibility to use Internet n-grams is limited because many queries to a search engine are slow and can lead to a ban for that search engine. Erroneous training data lead to incorrect n-grams in the database and these impair the grammar checking results. The problem that the sphere of a word is larger than the pentagram window can cover is only a minor issue in English. But in German the use of embedded sentences is common and thus there are quite often words whose sphere is larger than the pentagram window covers. This issue therefore depends on the used language.

The use of a tagset which contains tags for all necessary morphological features of the used language is recommended. The tagset needs to be chosen carefully. In general, a large tagset is better than a small one. However, a customized tagset for the actual usage helps to avoid unnecessary false positives. Furthermore, tagging accuracy is important to keep the false positives low. As the tagging accuracy cannot reach 100%, this issue needs to be accepted. The same applies for the usage of the tokenizer. The tokenizer is usually language dependent. We saw that at least the tokenizers for German and English are currently not capable of performing their task with 100% accuracy.

Some functionalities of LISGrammarChecker like the error threshold system, the correction proposal mechanism, and the agreement checks show in the real world test cases that they are not ready for usage yet in their current implementation. They need to be refined.

The statistical approach works in general, but some issues remain. Tests reveal that some issues can hardly be solved only with statistical data. This statistical approach depends very much on the statistical data itself. The individual n-gram checks have different advantages and disadvantages. A good error detection rate usually comes together with a high false positive rate. For example the token n-gram check detects most of the errors, but in return it causes many false positives if there are too few statistical data.

As we do not have enough statistical training data to ensure the proper working of the token n-gram check, we put our focus on the tag and hybrid n-gram checks. A combination of these two n-gram checks improves the results because the advantages of both checks are combined. To counteract the problem of false positives of the hybrid n-grams, we combine it with rules. Through the integration of rules, LISGrammarChecker indeed loses its language independence. But simultaneously it improves in tackling specific grammar problems. Thus the new program logic of LISGrammarChecker which combines the tag and hybrid n-gram checks with rules increases the accuracy of the overall results.

Concluding we can say that the statistical approach works, but depends on the quality and amount of the statistical data. Furthermore some flaws remain because of some external tools like the tagger and tokenizer which prevent unveiling the full power of the statistical approach. The best solution is thus a combination of rules within the statistical approach which improves the results.

10 Future work

This work can be extended in many ways. We propose improvements to the statistical approach itself so that the grammar checking is upgraded. There could also be other methods which improve the use of LISGrammarChecker and the user interaction. Other future work proposals extend LISGrammarChecker with more rules or combine it with existing programs.

10.1. More Statistical Data

LISGrammarChecker can be improved with better and more statistical data. The following could be done in order to improve the reliability of the statistical data and thus the checking.

Online database An online database is data storage which is available over the Internet. Every user can train and use it. Thus more statistical data is available for everyone. Wrong entries from one user can be eliminated by another. Thus, LISGrammarChecker can simply be used without time-consuming training in advance.

Google n-grams Google made available all its n-grams in 2006 [BF06]. This corpus does not consist of normal text but instead directly of about 1,1 billion pentagrams of tokens. To eliminate useless pentagrams, Google uses just sequences which occur more than 40 times. If this corpus is included in LISGrammarChecker, the statistical data would be much greater and thus grammar checking would be improved.

Hand corrected corpus Another improvement of the statistical data could be a hand correction of the Wortschatz Universität Leipzig corpora. The sentences itself and also the tokenization as well as the tags should be corrected and then trained to LISGrammarChecker.

10.2. Encoding

In general LISGrammarChecker can handle all available encodings on a certain system. At the moment the first input text to train the grammar checker specifies the encoding. A problem occurs if the grammar checker is then used with another text in a different encoding because at the moment there is no possibility implemented which distinguishes between different encodings. A solution for this problem is the re-encoding of the texts with Unicode (UTF-8). D offers possibilities to handle encodings, e.g. to verify the encoding or convert encodings (modules `std.utf` and `std.encoding`).

10.3. Split Long Sentences

One inconvenient issue of LISGrammarChecker and its main checking method is the handling of long sentences. Even grammar checkers based on rules have severe problems with long sentences because of their parsers which are faced with obstacles while building up the parse tree. Something near it applies for our approach. The sphere of a word is often larger than then biggest n-gram window (in our case a pentagram). Therefore it is not possible to check some grammatical structures for correctness. While the check for all tags of a sentence works well for short sentences, it is quite often wrong—and thus useless—for long sentences. The training corpus needs to be unlimited size to cover all possibilities of sentence structures. This goal is impossible to reach because—in many languages—sentences can have arbitrary length.

To solve this problem we think about an algorithm which splits long sentences into shorter sentence phrases. Because of language dependent punctuation, this algorithm is not language independent. It is based on rules how a sentence can be split. In general the rules are applied before any statistical approach is started.

The goal is to build up a tree where all leaves are separate complete sequences of text which can be checked by the n-gram logic. The following steps show in order what needs to be done.

1. The whole sentence is read.
2. The tokenization of the sentence is done. In addition of the tokenization into words, a special treatment for the captoids, factoids and multi word expressions is done. They are marked as one single token.
3. The tagger tags all tokens from the previous step. All punctuation gets a different and distinguishable tag.
4. A split on all semicolons is done. All resulting phrases are stored for further processing. The tree of the sentence is extended to track the structure of the sentence.
5. All generated parts from the last step are used and the direct speech is extracted. Therefore, the token sequence : " is used as marker for the start and " is used as the end marker of the direct speech. The tree of the sentence is updated.
6. The parts of the last step are regarded for indirect speech. Here the token " or the token sequence , " are used for the start marker. The end is marked by either " or " .. To be indirect speech, one of the markers must contain a comma. The tree of the sentence is updated once again.
7. All parts which are generated until now are split on all commas and the sentence tree gets new entries.

After these steps we have a tree of the sentence which shows the structure. Depending of the current mode, in which the grammar checker is executed, i.e. training or checking mode, the next steps differ. In training mode, the following steps are done:

1. All words of the sentence are stored in the database.
2. The sentence tree is stored in a distinct database table.
3. The tag sequences of the parts are stored in the database.
4. An n-gram analysis is done for all parts of the sentence. All features known from the grammar checker are used: token n-grams, tag n-grams and hybrid n-grams, all bi- up to pentagrams. The n-grams are put into the database.

In checking mode the available sentence parts need to be checked for their existence in the database. Furthermore, some rules can be applied. The following could be done:

1. All parts are checked using the n-gram check. Error points are set depending on the type of n-gram.

2. The tree is compared to the database. If it exists, it is considered as correct. If the structure is not found in the database, some rules can be used to verify the correctness of that structure.
3. Error proposals for a rearrangement of the parts (other tree structure) or for the individual parts can be derived from the stored entries in the database.

10.4. Statistical Information About Words and Sentences

The functionality of LISGrammarChecker could be extended with statistical information about words and sentences. For example the amounts of words in a sentence or the position of a word in a sentence can be used to perform further grammatical feature checks. One can think about words that have always a certain position relative to another in a sentence. An error could be assumed if that position differs from the expected one. Another error could be proposed, if the sentence length is much too long. For example if there are more than 40 words in a sentence, there could be a warning.

We already pinpoint unknown words in a sentence. At the moment these always result in an error. There could be a special treatment insofar that these errors are less weighted or further rules are applied.

10.5. Use n-gram Amounts

All amounts of n-grams are stored but we use them only for the correction proposal. Nevertheless, these amounts could help also to define the probability of each n-grams. Thus, if the probability of an n-gram is low because it is only rare this could be an indication that this n-gram is wrong or the word order is incorrect. This means instead that n-grams amounts that are below a threshold are treated as not found.

10.6. Include more Rules

In general, there are three possibilities to combine our statistical approach with rules.

1. Rules could be applied in advance of the statistical part. We have already given a proposal for this, i.e. the algorithm to split long sentences (section 10.3). Sentences are reduced to shorter phrases with the help of rules, then the statistical approach is performed.

2. Rules could be performed in parallel with the statistical methods. An example therefore is the rule-based combination of the different n-gram checks that we proposed above. Here, rules are applied to determine the weights of the different checking methods. Furthermore our approaches with the adverb-verb- and adjective-noun-agreements can considered as a combination in parallel of rules with statistics.

3. The statistical methods are applied first, and then rules are applied for further verification of the errors and improvements of the results. We have already implemented this procedure in LISGrammarChecker and we have seen that these rules help to solve issues with statistical grammar checking. The overall results have improved. Thus we propose to include more rules.

   Extension of current rules The already existing rules could be extended. Our prework makes this possible by simply inserting additional rules to the text file that contains the rules—as described in section 8.4.3.

   Range extension Currently we apply rules only to the n-grams which are pointed out as incorrect by LISGrammarChecker. This range could be extended to whole sentences. This means that rules which are used to verify the errors from the statistical approach could be applied to a whole sentence.

   Apply a parser The use of a parser could help to apply more complex rules. Let us consider the two phrases "...likes fish and Peter goes..." and "he and Peter go to...". Without the parsers knowledge about the sentence tree we cannot decide if there is a verb phrase (VP) or a noun phrase (NP) on the left side of the conjunction and. Although we have a pentagram in both example phrases, it is not possible to determine if the verb must be goes (3rd person singular) or go (plural) without a parser.

   Check correct sentences with rules At the moment we apply rules to the sentence phrases which are marked as potentially wrong by a statistical method. It could also be useful to apply rules to sentences where no errors are detected by the n-gram checks. This could help to detect errors which are not possible to find using just pentagrams.

10.7. Tagset that Conforms Requirements

A too small tagset causes many errors. This is the case for Penn Treebank tagset. A larger tagset, e.g. the Brown tagset, minimizes the amount of errors that are caused by the tagset. But the Brown tagset is still not perfect for our demands. There are e.g. many different classifications for nouns. The Brown tagset supports the distinction of normal, locative,

proper, temporal, and some more types of nouns. This sophisticated distinction is unnecessary to permit the n-gram checks and is even counterproductive. The possible tag sequence combinations are too many to be covered with the available statistical data and thus lead to false positives. The best solution would be a tagset that that conforms completely to our requirements, i.e. supports all needed features for the n-gram approach but not more.

10.8. Graphical User Interface

The communication between the user and LISGrammarChecker could be extended with a graphical user interface. This could increase the usability for several purposes and ease the use of our grammar checker. There are several possibilities how this could be realized. One way would be to write a frontend that uses a graphical interface, e.g. GTK. Another approach could be web-based. Furthermore, LISGrammarChecker could be combined with an existing text processing program, e.g. OpenOffice. For all approaches, the output of LISGrammarChecker would need to be converted to satisfy the interface to the frontend.

10.9. Intelligent Correction Proposal

In some cases it might not be sufficient to use only the pentagram with the middle word as wildcard for a correction proposal. This can happen if the error is at the beginning or at the end of a sentence and the error occurred at the first or last word of the sentence. Furthermore, it could be possible that not the word which is marked as wrong itself is wrong but also another adjacent word. For these cases the correction proposal should be extended. We propose the following possibilities:

Different positions of the wildcard Instead of setting the wildcard only in the center position, all tokens of the pentagram could be set as wildcard one after one. In the example "houses are usually vary secure" that means the first wildcard would replace the word houses, after that it would replace are and so on. The alternative which gives the highest amount in the database wins.

Swapping of two words Here the words are swapped with their right neighbor. Like in the previous step, there is only one swap at a time. All alternatives are compared to the database.

Insertion of a token A wildcard is set between two tokens. Then there are five tokens and one wildcard. To apply a pentagram search, one of the tokens on the left or right edge of the pentagram need to be skipped. This should be the one which is further away

from the wildcard. In the example "houses usually very secure ." it could be as follows: "houses * usually very secure" or "usually very * secure .".

Deletion of a word This works vice versa to to the insertion approach. Here, we think about six tokens in a row, where one word is skipped. The resulting pentagram is compared to the database.

Help of tag n-grams Instead of representing every token as a wildcard to search the database, the tag n-gram could be used. This could substitute the error token by the most probable token of a certain word class. In the above example augmented with tags "houses(NNS) are(VBP) usually(RB) vary(VBP) secure(JJ)". With a wildcard search of the most probable tag sequence we would find "(NNS) (VBP) (RB) (RB) (JJ)". That means we need to find an alternative for the word vary. It should be a word with the tag RB. A further search in the token n-grams shows that very has the tag RB and fits best.

Similarity search Instead of looking up the n-gram with the most occurrences, we propose to use the n-gram which is the most similar to the one searched.

Part V.

Appendix

Acronyms & Abbreviations

ANC American National Corpus

APSG Augmented Phrase Structure Grammar

BASH Bourne Again Shell

BNC British National Corpus

CFQ Completely Fair Queuing

GCC GNU Compiler Collection

GDC GNU D Compiler

HTML Hypertext Markup Language

LISGrammarChecker Language Independent Statistical Grammar Checker

NEGRA Nebenläufige Grammatische Verarbeitung

NLP Natural Language Processing

PHP PHP: Hypertext Preprocessor

PoS Part-of-Speech

POSIX Portable Operating System Interface

SQL Structured Query Language

stdin Standard Input

stdout Standard Output

STTS Stuttgart-Tübingen Tagset

TnT Trigrams'n'Tags

UTF-8 8-bit Unicode Transformation Format

XML Extensible Markup Language

B Glossary

American National Corpus This corpus aims to be a representation of American English and is currently build up.[ISO6]

Anaphora A referential pattern

Brown Corpus consists of approximately 1 million words of running text of edited English prose printed in the United States during the calendar year 1961. It is also denoted as the Standard Corpus of Present-Day American English. [FK64]

Captoid A multiword expression with all words capitalized, e.g. a title as "Language Independent Statistical Grammar Checker".

Combination algorithm A combination algorithm defines in which way different tag proposals from different taggers are combined so that there is exactly one (combined) tag for each word as result.

Combined tagging Combined tagging is a technology to improve the accuracy rate of a tagger. The system uses therefore two or more taggers with different technologies. All taggers make different mistakes and a combination is thus more precise than one of those taggers alone.

Corpus A corpus is a collection of written or spoken phrases that correspond to a specific natural language. The data of the corpus is typically digital, i.e. it is saved on computers and machine-readable. The corpus consists of the following components: The text data itself, possibly meta data which describes these text data, and linguistic annotations related to the text data.

Factoid Factoids are multiwords, for example dates or places.

Grammar A grammar of a natural language is a set of combinations (syntax) and modifications (morphology) of components and words of the language to form sentences.

Lexeme A synonym for token.

n-gram There are two types of n-grams—n-gram of tokens and n-gram of tags. An n-gram of tokens is a subsequence of neighbored tokens in a sentence. An n-gram of tags is a sequence of tags that describes such a subsequence of neighbored tokens in a sentence. In both cases, n defines the number of tokens.

NEGRA Corpus Corpus of Nebenläufige Grammatische Verarbeitung—an annotated corpus for the German language.

Part-of-speech denotes the linguistic category of a word.

Penn Treebank tagset One of the most important tagsets for the English language is built by the Penn Treebank Project [MSM93]. The tagset contains 36 part-of-speech tags and 12 tags for punctuation and currency symbols.

POSIX is the collective name of a family of related standards specified by the IEEE to define the application programming interface, along with shell and utilities interfaces for software compatible with variants of the Unix operating system, although the standard can apply to any operating system.

Standard input/output This denotes the standard streams for input and output in POSIX compatible operating system

Stuttgart-Tübingen Tagset A tagset for the German language which consists of 54 tags. [STST99]

Tag The morphosyntactic features that are assigned to tokens during tagging are represented as strings. These strings are denoted as tags.

Tagger Performs the task of tagging.

Tagging The process of assigning the word class and morphosyntactic features to tokens is called tagging.

Tagging accuracy The tagging accuracy is measured as the number of correctly tagged tokens divided by the total number of tokens in a text.

Tagset The set of all tags constitutes a tagset.

Token Every item in a text is a token, e.g. words, numbers, punctuations, or abbreviations.

Tokenization is the breaking down of text to meaningful parts like words and punctuation, i.e. the segmentation of text into tokens.

C Eidesstattliche Erklärung

Wir versichern hiermit, daß wir die vorliegende Arbeit selbständig verfaßt und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von uns selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Die Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, den 20. Februar 2009

Verena Henrich

Timo Reuter

Bibliography

D

[ABC] ABCNews Internet Ventures : abc NEWS. — http://abcnews.go.com/ Accessed 15-12-2008

[AUK06] Alam, Md. J. ; UzZaman, Naushad ; Khan, Mumit: N-gram based Statistical Grammar Checker for Bangla and English. In: Proceedings of ninth International Conference on Computer and Information Technology (ICCIT 2006). Dhaka, Bangladesh, 2006. — http://www.panl10n.net/english/finalsh/BAN21.pdf

[Bat94] Batstone, Rob: Grammar: A Scheme for Teacher Education. Oxford University Press, 1994. — ISBN 0194371328. — http://books.google.de/books?id=0TWje50dS1EC

[BF06] Brants, Thorsten ; Franz, Alex: Web IT 5-gram Version 1. Philadelphia, USA : Linguistic Data Consortium, 2006. — Catalog No. LDC2006T13, ISBN 1-58563-397-6, Release Date 19-09-2006. http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T13

[BHK+97] Brants, Thorsten ; Hendriks, Roland ; Kramp, Sabine ; Krenn, Brigitte ; Preis, Cordula ; Skut, Wojciech ; Uszkoreit, Hans: Das NEGRA-Annotationsschema / Universität des Saarlandes. Saarbrücken, Germany, 1997. — Negra Project Report. — http://www.coli.uni-sb.de/sfb378/negra-corpus/negra-corpus.html

[Bis05] Bishop, Todd: A Word to the unwise – program's grammar check isn't so smart. 2005. — Online article in Seattle Post-Intelligencer. http://seattlepi.nwsource.com/business/217802_grammar28.asp Last modified 28-03-2005. Accessed 11-01-2009

[Blo] Bloomberg L.P.: Bloomberg.com. — http://www.bloomberg.com/ Accessed 15-12-2008

[Bra00] Brants, Thorsten: TnT — A Statistical Part-of-Speech Tagger. In: Proceedings of the Sixth Applied Natural Language Processing (ANLP-2000). Seattle, WA, 2000, pp. 224—231. — http://www.coli.uni-saarland.de/~thorsten/publications/Brants-ANLP00.pdf

[Bri94] Brill, Eric: Some Advances in Transformation-Based Part of Speech Tagging. In: Proceedings of AAAI, Vol. 1, 1994, pp. 722—727. — http://www.aaai.org/Papers/AAAI/1994/AAAI94-110.pdf

[Bur07] Burnard, Lou: Reference Guide for the British National Corpus (XML Edition) / Published for the British National Corpus Consortium by the Research Technologies Service at Oxford University Computing Services. 2007. — Technical Report. — http://www.natcorp.ox.ac.uk/XMLedition/URG/

[Cor] Corel Corporation: WordPerfect Office. — http://www.corel.com/servlet/Satellite/us/en/Product/1207676528492#tabview=tab0 Accessed 15-01-2009

[Dig] Digital Mars: D Programming Language. — http://www.digitalmars.com/d/ Accessed 15-01-2009

[DMS00] George E. Heidorn: Intelligence Writing Assistance. In: Dale, R. ; Moisl, H. ; Somers, H.: A Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text. New York, USA : Marcel Dekker, 2000. — ISBN 3823362100, pp. 181—207

[FK64] Francis, W. N. ; Kucera, H. ; Department of Linguistics, Brown University (eds.): BROWN CORPUS MANUAL – MANUAL OF INFORMATION to accompany a Standard Corpus of Present-Day Edited American English, for use with Digital Computers. Providence, Rhode Island: Department of Linguistics, Brown University, 1964. — Revised and Amplified 1979. http://khnt.hit.uib.no/icame/manuals/brown/INDEX.HTM

[Fos04] Foster, Jennifer: Good Reasons for Noting Bad Grammar: Empirical Investigations into the Parsing of Ungrammatical Written English. Dublin, Ireland, Department of Computer Science, Trinity College, University of Dublin, Diss., 2004. — http://www.cs.tcd.ie/research_groups/clg/Theses/jfoster.ps

[Fri] Friedmann, David: GDC – D Programming Language for GCC. — http://dgcc.sourceforge.net/ Accessed 15-01-2009

[Gooa] Google Inc.: Google™. — http://www.google.com/ Accessed 15-01-2009

[Goob] Google Inc.: Google™Scholar BETA. — http://scholar.google.com/ Accessed 15-01-2009

[Hil] Hillyer, Mike: An Introduction to Database Normalization. — http://dev.mysql.com/tech-resources/articles/intro-to-normalization.html

[Hol04] Sigrún Helgadóttir: Testing Data-Driven Learning Algorithms for PoS Tagging of Icelandic. In: Holmboe, H.: Nordisk Sprogteknologi 2004 – Nordic Language Technology – Årbog for Nordisk Sprogteknologisk Forskningsprogram 2000-2004. Kopenhagen, Denmark : Museum Tusculanum Press, 2004. — ISBN 9788763502481

[HZD01] Halteren, Hans van ; Zavrel, Jakub ; Daelemans, Walter: Improving Accuracy in Wordclass Tagging through Combination of Machine Learning Systems. In: Computational Linguistics 27 (2001), No. 2, pp. 99—230. — http://www.cnts.ua.ac.be/Publications/2001/HZD01/20010718.7496.hzd01.pdf

[IEE04] IEEE: IEEE Std 1003.1, 2004 Edition, Single UNIX Specification Version 3. 2004. — Institute of Electrical and Electronics Engineers, Inc. and Open Group. http://www.unix.org/version3/ieee_std.html

[IS06] Ide, Nancy ; Suderman, Keith: Integrating Linguistic Resources: The American National Corpus Model. In: Proceedings of the Fifth Language Resources and Evaluation Conference (LREC). Genoa, Italy, 2006. — http://www.cs.vassar.edu/-ide/papers/ANC-LREC06.pdf

[Kie08] Kies, Daniel: Evaluating Grammar Checkers: A Comparative Ten-Year Study. 2008. — In: The HyperTextBooks — Modern English Grammar — English 2126. Department of English, College of DuPage. Website http://papyr.com/hypertextbooks/grammar/gramchek.htm Last modified 27-12-2008. Accessed 11-01-2009

[KP96] Kernick, Philip S. ; Powers, David M.: A Statistical Grammar Checker. Adelaide, South Australia, Aug 1996. — Department of Computer Science, Flinders University of South Australia, Honours Thesis. http://david.wardpowers.info/Research/AI/papers/199608-sub-SGC.pdf

[KY94] Kantz, Margaret ; Yates, Robert: Whose Judgments? A Survey of Faculty Responses to Common and Highly Irritating Writing Errors. Warrensburg, MO, USA, Aug 1994. — A paper presented at the Fifth Annual Conference of the NCTE Assembly for the Teaching of English Grammar, Illinois State University. http://www.ateg.org/conferences/c5/kantz.htm

[Lana] LanguageTool: LanguageTool – Development. — http://www.languagetool.org/development/#process Last modified 11-10-2008. Accessed 11-01-2009

[Lanb] LanguageTool: LanguageTool – Open Source language checker. — http://www.languagetool.org/ Last modified 11-10-2008. Accessed 11-01-2009

[Lina] Linguisoft Inc.: Grammarian Pro X. — http://linguisoft.com/gramprox.html Accessed 11-01-2009

[Linb] Linguistic Data Consortium: Linguistic Data Consortium. — University of Pennsylvania. http://www.ldc.upenn.edu/ Last modified 08-01-2009. Accessed 13-01-2009

[Lof07] Loftsson, Hrafn: The Icelandic tagset. Department of Computer Science, Reykjavik University, Reykjavik, Iceland, Jan 2007. — http://nlp.ru.is/pdf/Tagset.pdf

[Lof08] Loftsson, Hrafn: Tagging Icelandic text: A linguistic rule-based approach. In: Nordic Journal of Linguistics, Cambridge University Press, 2008, pp. 47—72. — http://www.ru.is/faculty/hrafn/Papers/IceTagger_final.pdf

[LZ06] Lemnitzer, Lothar ; Zinsmeister, Heike: Korpuslinguistik: Eine Einführung. Gunter Narr Verlag, 2006. — ISBN 3823362100. — http://books.google.com/books?id=Lxe2aO9dwoAC&hl=de

[Mica] Microsoft: Microsoft(R). — http://www.microsoft.com/ Accessed 11-01-2009

[Micb] Microsoft Corporation: Microsoft(R) Office Online – Microsoft Office Word. — http://www.microsoft.com/office/word Accessed 11-01-2009

[MS] MySQL AB ; Sun Microsystems, Inc.: MySQL. — http://www.mysql.com/ Accessed 02-01-2009

[MSM93] Marcus, Mitchell P. ; Santorini, Beatrice ; Marcinkiewicz, Mary A.: Building a Large Annotated Corpus of English: The Penn Treebank / Department of Computer and Information Science, University of Pennsylvania. 1993 (MS-CIS-93-87). — Technical Report. — http://repository.upenn.edu/cgi/viewcontent.cgi?article=1246&context=cis_reports

[Ope] OpenOffice.org: OpenOffice.org – The free and open productivity suite. — http://www.openoffice.org/ Accessed 11-01-2009

[Pen] Penn Treebank Project: Treebank tokenization. — Computer and Information Science Department, University of Pennsylvania. http://www.cis.upenn.edu/~treebank/tokenization.html

[PMB91] Pind, Jörgen ; Magnússon, Friðrik ; Briem, Stefán: Íslensk Orðtíðnibók (Frequency Dictionary of Icelandic). Reykjavik, Iceland : The Institute of Lexicography, University of Iceland, 1991

[Pöh] Pöhland, Jörg: englisch-hilfen.de – Learning English Online. — http://www.englisch-hilfen.de/en/

[QH06] Quasthoff, Uwe ; Heyer, Gerhard ; Natural Language Processing Department, University of Leipzig (eds.): Leipzig Corpora Collection User Manual – Version 1.0. Stuttgart, Germany: Natural Language Processing Department, University of Leipzig, May 2006. — http://corpora.informatik.uni-leipzig.de/download/LCCDoc.pdf

[Sch94] Schmid, Helmut: Probabilistic Part-of-Speech Tagging Using Decision Trees. In: Proceedings of the International Conference on New Methods in Language Processing. Stuttgart, Germany, 1994. — http://www.ims.uni-stuttgart.de/ftp/pub/corpora/tree-tagger1.pdf

[Sch00] Schmid, Helmut: Unsupervised Learning of Period Disambiguation for Tokenisation / Institute for Natural Language Processing, University of Stuttgart. 2000. — Internal Report. — http://www.ims.uni-stuttgart.de/~schmid/tokeniser.pdf

[Sjö03] Sjöbergh, Jonas: Combining POS-taggers for improved accuracy on Swedish text. In: Proceedings of the 14th Nordic Conference of Computational Linguistics (NoDaLiDa 2003). Reykjavik, Iceland, 2003. — http://dr-hato.se/research/combining03.pdf

[Ste03] Steiner, Petra: Das revidierte Münsteraner Tagset/Deutsch (MT/D) — Beschreibung, Anwendung, Beispiele und Problemfälle / Arbeitsbereich Linguistik, University of Münster. 2003. — Technical Report. — http://santana.uni-muenster.de/Publications/tagbeschr_final.ps

[STST99] Schiller, Anne ; Teufel, Simone ; Stückert, Christine ; Thielen, Christine: Guidelines für das Tagging deutscher Textcorpora mit STTS (Kleines und großes Tagset) / Institute for Natural Language Processing, University of Stuttgart and Department of Linguistics, University of Tübingen. 1999. — Technical Report. — http://www.ifi.uzh.ch/~siclemat/man/SchillerTeufel99STTS.pdf

[The] The New York Times Company: The New York Times. — http://www.nytimes.com/ Accessed 15-12-2008

[TM00] Toutanova, Kristina ; Manning, Christopher D.: Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. In: Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000). Hong Kong, China, 2000, pp. 63—70. — http://nlp.stanford.edu/~manning/papers/emnlp2000.pdf

[VOA] VOANews.com: VOANews.com – Voice of America. — http://www.voanews.com/ Accessed 15-12-2008

[Yah] Yahoo! Inc.: Yahoo!®. — http://www.yahoo.com/ Accessed 15-01-2009

[Yer03] Yergeau, F.: UTF-8, a transformation format of ISO 10646. Nov 2003. — Request for Comments 3629. http://tools.ietf.org/rfc/rfc3629.txt

E Resources

In this part of the appendix, we list resources that we implemented relating to the development of LISGrammarChecker. We provide several listings that show implementation details from our grammar checker. Finally, our self-made error corpora are shown. The main reason for this chapter is to provide our resources to others. While we worked on this thesis, we often found other documentations where we could not get the resources anymore. To avoid this with our resources, we include them in our documentation.

E.1. Listings

We provide the implementation of our simple voting combination algorithm and our possibility to call extern programs in D.

E.1.1. Simple Voting Algorithm

This listing shows the implementation of our simple voting algorithm. It consists of function `simpleVoting` which is found in module `taggers.tagger`.

```
/**
 * Do simple voting on the evaluated array using all taggers input. Result is written
 * back to the appropriate fields in the evaluated array.
 * Params:
 *     (inout) evaluated_lexemes_lemmas_tags = Evaluated array.
 */
void simpleVoting(inout char[][][] evaluated_lexemes_lemmas_tags)
{
    char[][] current_tag;
    int[] current_tag_count;
    int k, temp_highest_value;
    bool tag_not_found;

    // Go through all lexemes and tags
    for (int i = 0; i < evaluated_lexemes_lemmas_tags.length; i++)
    {
        current_tag.length = 1;
        current_tag_count.length = 1;
```

```
19          int k = -1;
20          bool tag_not_found = true;
21
22          // Go through all tags from the different taggers
23          for (int j = 3; j < evaluated_lexemes_lemmas_tags[0].length; j++)
24          {
25              for (int l = 0; l <= k; l++)
26              {
27                  if (cmp(evaluated_lexemes_lemmas_tags[i][j], current_tag[l]) == 0)
28                  {
29                      current_tag_count[l]++;
30                      tag_not_found = false;
31                  }
32                  else
33                  {
34                      tag_not_found = true;
35                  }
36              }
37              if (tag_not_found)
38              {
39                  current_tag.length = current_tag.length + 1;
40                  current_tag_count.length = current_tag_count.length + 1;
41                  k++;
42                  current_tag[k] = evaluated_lexemes_lemmas_tags[i][j];
43                  current_tag_count[k] = 1;
44              }
45          }
46
47          temp_highest_value = 0;
48
49          // Go through all tags for one lexeme
50          for (int l = 0; l < current_tag.length; l++)
51          {
52              if (temp_highest_value < current_tag_count[l])
53              {
54                  temp_highest_value = current_tag_count[l];
55                  evaluated_lexemes_lemmas_tags[i][2] = current_tag[l];
56              }
57          }
58      }
59  }
```

Listing E.1: Simple voting algorithm

E.1.2. Shell Function to Call Extern Programs

This listing is the implementation of our `shell` function in module `standard.shell`. It can be used to call extern programs from a D program. See section 6.4 for further explanations.

```
1  char[] shell(char[] cmd)
2  {
3      FILE* f = popen(toStringz(cmd), toStringz("r"));
4      char[] line;
5      char[] result;
6      while (readln(f, line))
7      {
8          result ~= line;
```

```
    }
    fclose(f);
    return result;
}

extern(C) FILE* popen(char* name, char* mode);
```

Listing E.2: Shell function

E.2. Error Corpora

Here we provide our self-made error corpora. They are all in the following format:

- Every line starts with a letter, either A, B or C. Letter A marks intentionally wrong sentences. Letter B marks the corrected version of the same sentence. In some cases there exists also a variant C, which denotes a second correct alternative of the same sentence.
- A period and a space follow to distinguish between the type marker and the sentence itself.
- Finally the line contains the sentence itself.

E.2.1. Self-made Error Corpus (English)

This error corpus is made up on our own. Therefore we have randomly selected sentence parts from the Wortschatz Universität Leipzig corpus [QH06] to form new sentences. Now it consists of newly created English sentences. The wrong sentences contain grammatical errors from chapter 2.1.4.

```
A. A statement for a said the rebel forces in Manhattan manage a 46 second showing.
B. A statement for a club said the rebel forces in Manhattan manage a 46 second showing.
A. The president remarked that that he likes the wonderful job.
B. The president remarked that he likes the wonderful job.
A. The official did not knowing why the decision was made.
B. The official did not know why the decision was made.
A. Towards the end both sides said the study assumes that the are most active.
B. Towards the end both sides said the study assumes that the Japanese are most active.
A. The combination of the idea has been uses over the holiday weekend.
B. The combination of the idea has been used over the holiday weekend.
A. He has done great job in the private sector.
B. He has done a great job in the private sector.
A. The sixth-largest banking group in the United States must resolve these money problem before
      the general elections next year.
B. The sixth-largest banking group in the United States must resolve this money problem before
     the general elections next year.
A. After move to South Korea, which is not part of it, we became farmers.
B. After moving to South Korea, which is not part of it, we became farmers.
```

```
A. This is the first time moved to a house outside Washington.
B. This is the first time he moved to a house outside Washington.
A. The will try to play at the top of the hills.
B. They will try to play at the top of the hills.
A. Ironically, many stores make spot checks of professional actors, and the main reason for
   that is that the stock close at $4,700 a share.
B. Ironically, many stores make spot checks of professional actors, and the main reason for
   that is that the stock closed at $4,700 a share.
A. The rates were the high since Sept. 5.
B. The rates were the highest since Sept. 5.
A. The earlier negotiations have to go on a waiting list or your certainly aren't going to set
   the tone for next week.
B. The earlier negotiations have to go on a waiting list or you certainly aren't going to set
   the tone for next week.
A. I imagine he said the agreement what reached by a representative of the Prime Minister.
B. I imagine he said the agreement was reached by a representative of the Prime Minister.
A. Within minutes, she promised they the crowd that no new major economic initiatives are
   likely.
B. Within minutes, she promised the crowd that no new major economic initiatives are likely.
A. Midsize firms assume that individuals seek to maximize satisfaction while businessmen
   maximizes profits.
B. Midsize firms assume that individuals seek to maximize satisfaction while businessmen
   maximize profits.
A. I can tell you when they cite the studies that have been done they don't include his very
   often.
B. I can tell you when they cite the studies that have been done, they don't include his very
   often.
A. Of that group, only people who have quite a lot of our quasi-capital are up or election this
    year.
B. Of that group, only people who have quite a lot of our quasi-capital are up for election
   this year.
A. After the operation, he to confirm that immunity had been granted to NASA officials.
B. After the operation, he refused to confirm that immunity had been granted to NASA officials.
A. I'm am not paying the neccesary attention to them.
B. I'm not paying the neccesary attention to them.
A. The water pipes where broken and the party died Saturday at age 55.
B. The water pipes were broken and the party died Saturday at age 55.
A. Mathematic made little difference.
B. Mathematics made little difference.
A. These also find favor with girls whose believe that Joan died from a heart attack.
B. These also find favor with girls who believe that Joan died from a heart attack.
A. Another 20 homes are entitled too a receipt for the money.
B. Another 20 homes are entitled to a receipt for the money.
A. It could of run out of minivans because the two plants that build the models will be closed
   for four weeks.
B. It could run out of minivans because the two plants that build the models will be closed for
    four weeks
A. Before that, there competitors had been fighting since 1981.
B. Before that, their competitors had been fighting since 1981.
A. In the past week, he say he agreed because they see themselves as Bosnian.
B. In the past week, he said he agreed because they see themselves as Bosnian.
A. But he says the situation there has not changed for a decade much.
B. But he says the situation there has not changed much for a decade.
A. He was unavailable with in his private law office Thursday morning.
B. He was unavailable in his private law office Thursday morning.
A. Mostly, takeover speculation has nothing to do to the company.
B. Mostly, takeover speculation has nothing to do with the company.
A. The act of a military provocation like this is planned is planned.
B. The act of a military provocation like this is planned.
A. The exchanges in Washington already deciding that it wasn't a good place to invest.
B. The exchanges in Washington were already deciding that it wasn't a good place to invest.
```

```
A. According to one conference organizer, is is both an agreement and it's been thoroughly
   thought through.
B. According to one conference organizer, it is both an agreement and it's been thoroughly
   thought through.
A. NBC's loss is partly due to fact that the baseball playoffs did not go for a sixth or
   seventh game.
B. NBC's loss is partly due to the fact that the baseball playoffs did not go for a sixth or
   seventh game.
A. Many people were absent, finishing touches no its minimum wage bill.
B. Many people were absent, finishing touches on its minimum wage bill.
A. Since then, Jim in been giving officials free trips to the southern border.
B. Since then, Jim has been giving officials free trips to the southern border.
A. Analysts said Australia has awarded too licenses within the last three years.
B. Analysts said Australia has awarded two licenses within the last three years.
A. Him is giving up the Daily Star.
B. He is giving up the Daily Star.
A. Fewer than one-fourth from those asked will come out of the investment banking business.
B. Fewer than one-fourth of those asked will come out of the investment banking business.
A. The plans was to differ on just about every issue.
B. The plan was to differ on just about every issue.
A. You enter the little auberge and its that complicated and that simple.
B. You enter the little auberge and it's that complicated and that simple.
A. Moreover he says officials want to transfer small objects or packages for currency in a
   furtive fashion.
B. Moreover, he says officials want to transfer small objects or packages for currency in a
   furtive fashion.
A. The long term goal must be to demand change or it amounts to national disease.
B. The long term goal must be to demand change or it amounts to a national disease.
A. He tells of the son of city Water Bureau official.
B. He tells of the son of a city Water Bureau official.
A. And in Washington, it became a tide sweeping form east to west after a joint global sourcing
    plan collapsed a year ago.
B. And in Washington, it became a tide sweeping from east to west after a joint global sourcing
    plan collapsed a year ago.
A. Police said shares of Southern Glass will be available to foreigner's by the and of the year
   .
B. Police said shares of Southern Glass will be available to foreigners by the and of the year.
A. When he finally reaches his coronation, plenty of risk there is in the car business.
B. When he finally reaches his coronation, there is plenty of risk in the car business.
A. The jury trial concludes that virtual all asthma attacks are triggered by the upper house.
B. The jury trial concludes that virtually all asthma attacks are triggered by the upper house.
A. After seven years later, above the steady gurgle of the river, a splash is preceded.
B. Seven years later, above the steady gurgle of the river, a splash is preceded.
A. The company said at least one speaker was taken away of the students.
B. The company said at least one speaker was taken away by the students.
C. The company said at least one speaker of the students was taken away.
A. Anchorman Tom Brokaw has found guilty of using drugs.
B. Anchorman Tom Brokaw has been found guilty of using drugs.
A. The pilot reported the engine failure and an veteran was sent to meet the plane.
B. The pilot reported the engine failure and a veteran was sent to meet the plane.
A. Biotechnology already makes it feasible home tests for a wide range of diseases.
B. Biotechnology already makes feasible home tests for a wide range of diseases.
A. They show also stars Mr. Flake.
B. The show also stars Mr. Flake.
A. They have have told the Poles that its offices have been visited by police looking for
   information.
B. They have told the Poles that its offices have been visited by police looking for
   information.
A. He may have been qualified to president of the country.
B. He may have been qualified to be president of the country.
A. I was thinking to more about operating cash flow.
B. I was thinking more about operating cash flow.
```

```
A. It is important but to complex too explain.
B. It is important but too complex to explain.
A. The rumors hung in Iraq.
B. The rumors hung over Iraq.
A. The other driver puts the spotlight on sizeable community in the Miami area.
B. The other driver puts the spotlight on the sizeable community in the Miami area.
A. They are very religious and the influence is society is high.
B. They are very religious and the influence in society is high.
A. He also worked at airport.
B. He also worked at the airport.
A. We have going to have to deal with the problems of segregated schools.
B. We are going to have to deal with the problems of segregated schools.
A. A Danish statement said they continue as treasurer a member of the operating committee for
   this travel company.
B. A Danish statement said they continue as treasurer and a member of the operating committee
   for this travel company.
A. He was also own for holding more than 11% of the home market.
B. He was also known for holding more than 11% of the home market.
A. Later, she recommends a salary audit later every three years.
B. Later, she recommends a salary audit every three years.
A. Anyone over the age of 18 can wins the game.
B. Anyone over the age of 18 can win the game.
A. Each years the costs for roads can barely raise the $52 million.
B. Each year the costs for roads can barely raise the $52 million.
A. A Colombian coffee official apparent realized he would not be released.
B. A Colombian coffee official apparently realized he would not be released.
A. There is a obligation to negotiate.
B. There is an obligation to negotiate.
A. Yesterday, it has gone very well.
B. Yesterday, it went very well.
A. The one who signed price stability contract with the government, agreed his job was on the
   line.
B. The one who signed a price stability contract with the government, agreed his job was on the
    line.
A. These groups want Jim allow a multiparty system.
B. These groups want Jim to allow a multiparty system.
A. I have no idea to what they're talking about.
B. I have no idea what they're talking about.
A. Readers interested in general references grammars of English should waste no time.
B. Readers interested in general reference grammars of English should waste no time.
A. The user will interact will Jim's new computer.
B. The user will interact with Jim's new computer.
A. They work on those who voted against the leaders wishes.
B. They work on those who voted against the leaders' wishes.
A. The decision came of a meeting Tuesday between the president and his advisors.
B. The decision came out of a meeting Tuesday between the president and his advisors.
A. He describes himself as a journalist with on a mission.
B. He describes himself as a journalist with a mission.
C. He describes himself as a journalist on a mission.
A. Since then, they have lay untouched.
B. Since then, they have lain untouched.
A. I have no objection to a bank applying any interest rate at all as as long as they state it
   in a way that allows you to compare accounts.
B. I have no objection to a bank applying any interest rate at all as long as they state it in
   a way that allows you to compare accounts.
A. Scientists reported that their solar collector is capable to concentrate sunlight.
B. Scientists reported that their solar collector is capable of concentrating sunlight.
A. But most families that turn the public sector find ways to be happy.
B. But most families that turn to the public sector find ways to be happy.
A. All workers at his ministry will be subjected to the tithing policy as July 1.
B. All workers at his ministry will be subjected to the tithing policy as of July 1.
A. But one of the good thing about it is that it isn't very good.
```

```
172 B. But one of the good things about it is that it isn't very good.
173 A. He was conscious but he unable to speak during Wednesday's arraignment.
174 B. He was conscious but unable to speak during Wednesday's arraignment.
175 A. There are already signs that the recently raise 41% rate will be a problem.
176 B. There are already signs that the recently raised 41% rate will be a problem.
177 A. His portfolio comprises of more than 3,000 episodes of movies.
178 B. His portfolio comprises more than 3,000 episodes of movies.
179 C. His portfolio consists of more than 3,000 episodes of movies.
180 A. He had to refuse the invitations because of only a few are allowed to travel.
181 B. He had to refuse the invitations because only a few are allowed to travel.
182 A. She may have missed a trick by not putting this programmes first.
183 B. She may have missed a trick by not putting this programme first.
184 A. It will prompt to the surgeon for advice.
185 B. It will prompt the surgeon for advice.
186 A. Some firms declined to help underwriting the offering.
187 B. Some firms declined to help underwrite the offering.
188 A. White House will find out that this policy will lead them nowhere.
189 B. The White House will find out that this policy will lead them nowhere.
190 A. In both cases, part of the reason of the legislation was to allay the public's fear.
191 B. In both cases, part of the reason for the legislation was to allay the public's fear.
192 A. We don't want to stop been a developer, but we will be more careful.
193 B. We don't want to stop being a developer, but we will be more careful.
194 A. Higher earnings are expected over next two years for the company.
195 B. Higher earnings are expected over the next two years for the company.
196 A. The whole thing makes no sentence to me.
197 B. The whole thing makes no sense to me.
198 A. His 14-year-old daughter was going the trip.
199 B. His 14-year-old daughter was going on the trip.
200 A. She asks departments whether they think the policy is giving them benefits and don't even
      have a fully working telephone.
201 B. She asks departments whether they think the policy is giving them benefits and doesn't even
      have a fully working telephone.
202 A. A computer hacker, who send messages to the narrowly defined audiences, cannot be prosecuted
      .
203 B. A computer hacker, who sent messages to the narrowly defined audiences, cannot be prosecuted
      .
204 A. It could set a positive tone if they are favorable but touch off a dollar crisis it they
      aren't.
205 B. It could set a positive tone if they are favorable but touch off a dollar crisis if they
      aren't.
206 A. Peter and Jim are looking for to make fast trading profits.
207 B. Peter and Jim are looking to make fast trading profits.
208 A. We give thanks to Got that this was invented.
209 B. We give thanks to God that this was invented.
210 A. At least two rider required hospitalization after falling from high windows.
211 B. At least two riders required hospitalization after falling from high windows.
212 A. It points to a links between the best-known observer groups and the leftist Party of
      Democratic Revolution as evidence of possible bias.
213 B. It points to the links between the best-known observer groups and the leftist Party of
      Democratic Revolution as evidence of possible bias.
214 A. But it appears to be on a steady downward trend, suggests that the drachma will be able to
      join the wide band of the exchange rate mechanism.
215 B. But it appears to be on a steady downward trend, suggesting that the drachma will be able to
       join the wide band of the exchange rate mechanism.
216 A. The ranking GOP member on the budget committee has selected.
217 B. The ranking GOP member on the budget committee has been selected.
218 A. The refugees rise fears of an uncontrollable flood from Southeast Asia.
219 B. The refugees raise fears of an uncontrollable flood from Southeast Asia.
220 A. Its an overall credit-rating decision.
221 B. It's an overall credit-rating decision.
222 A. Ryan was stinged by a bee as they ran through the cornfield from the plane.
223 B. Ryan was stung by a bee as they ran through the cornfield from the plane.
```

```
A. So your off on holiday this week and you haven't had a moment to think about the paperbacks.
B. So you're off on holiday this week and you haven't had a moment to think about the
   paperbacks.
A. Two principal shareholders wanted to sell there combined 42% stake.
B. Two principal shareholders wanted to sell their combined 42% stake.
A. They want to know as to why they aren't sleeping, why they want to throw up when they eat.
B. They want to know why they aren't sleeping, why they want to throw up when they eat.
A. She and Lorin is more than willing to lend if they can find worthy borrowers.
B. She and Lorin are more than willing to lend if they can find worthy borrowers.
A. Then we could of celebrate the new year on an agreed first day of Spring.
B. Then we could celebrate the new year on an agreed first day of Spring.
A. The chancellor has been at the scene of to many accidents.
B. The chancellor has been at the scene of too many accidents.
A. One area were the Gulf states do seem united is in their changed relations with outside
   states.
B. One area where the Gulf states do seem united is in their changed relations with outside
   states.
A. Mr. Hodel also raised concerns that the U.S. might commit themselves to an ineffective
   international treaty.
B. Mr. Hodel also raised concerns that the U.S. might commit itself to an ineffective
   international treaty.
A. And she said Tuesday she was not sure how her would vote.
B. And she said Tuesday she was not sure how she would vote.
A. The shares have fallen this far they seldom come back.
B. The shares have fallen this far, they seldom come back.
A. I did not reject the powerful influences of Europe on America, but I ask that the
   contributions of Africa, Asia and South America be granted the light of day in America's
   classrooms.
B. I do not reject the powerful influences of Europe on America, but I ask that the
   contributions of Africa, Asia and South America be granted the light of day in America's
   classrooms.
A. She looked right at me and she smiles broadly.
B. She looked right at me and she smiled broadly.
A. The company said it's rate of sales growth for the quarter has slowed from the 33% pace
   during the second quarter.
B. The company said its rate of sales growth for the quarter has slowed from the 33% pace
   during the second quarter.
A. I promise if somebody starts playing fast and lose with the university, they'll have to
   answer.
B. I promise if somebody starts playing fast and loose with the university, they'll have to
   answer.
A. After reading the original study, the article remains unconvincing.
B. After reading the original study, I find the article unconvincing.
A. It is nearly half past five, we cannot reach town before dark.
B. It is nearly half past five and we cannot reach town before dark.
A. One not entirely accidental side affect of the current crackdown will be a dampening of the
   merger and acquisition boom.
B. One not entirely accidental side effect of the current crackdown will be a dampening of the
   merger and acquisition boom.
A. He and its wife, Theda, wanted to stay in Atlanta.
B. He and his wife, Theda, wanted to stay in Atlanta.
A. Yesterday, they create 33 new ones.
B. Yesterday, they created 33 new ones.
A. I never worked harder in my life then in the last couple of years.
B. I never worked harder in my life than in the last couple of years.
A. His defeated challenger must remain in Washington answering slander charges.
B. His defeated challenger must remain in Washington to answer slander charges.
```

Listing E.3: Self-made Error Corpus (English)

E.2.2. Self-made Error Corpus with Simple Sentences (English)

We provide a small error corpus with just simple sentences. These sentences are made up by our own.

```
A. Max driver the car carefully.
B. Max drives the car carefully.
A. Here's is Peter.
B. Here is Peter.
A. My grandmother was joke.
B. My grandmother was joking.
A. She is watches TV.
B. She watches TV.
A. The dog has gets angry.
B. The dog gets angry.
A. Does the cat play in garden?
B. Does the cat play in the garden?
A. Is she is visiting the Tower of London?
B. Is she visiting the Tower of London?
A. The childrens love tennis, but they do not like biscuits.
B. The children love tennis, but they do not like biscuits.
A. They rides their bikes to the town.
B. They ride their bikes to the town.
A. I speak Germany.
B. I speak German.
A. If Maria call her friend, she begins with the game.
B. If Maria calls her friend, she begins with the game.
A. Alex could understand the text if he at it again.
B. Alex could understand the text if he looked at it again.
A. Houses is usually not expensive.
B. Houses are usually not expensive.
A. The boy drives an blue car.
B. The boy drives a blue car.
A. Doris is jumping over wall.
B. Doris is jumping over the wall.
A. Three car need to be repaired after they have crashed at the wall.
B. Three cars need to be repaired after they have crashed into the wall.
A. The police had problems in is writing the report.
B. The police had problems in writing the report.
A. It is looking greatly!
B. It is looking great!
A. Nina gets by the car and drives to school.
B. Nina gets on the car and drives to school.
A. Karen and Dave will arrives tomorrow.
B. Karen and Dave will arrive tomorrow.
A. A lots of people are angry about the farmers.
B. A lot of people are angry about the farmers.
A. A computer can be controlled by clicking an mouse button.
B. A computer can be controlled by clicking a mouse button.
A. The President works at the white House.
B. The President works at the White House.
A. This problem definitely is happened.
B. This problem definitely happened.
A. Republicans have a differently approach.
B. Republicans have a different approach.
A. Democrats had previously expressed a fear that the price is high is.
B. Democrats had previously expressed a fear that the price is high.
A. The men was too tall.
B. The man was too tall.
A. The unemployments rate jumps.
```

```
B. The unemployment rate jumps.
A. They the will invest in large cattle ranches.
B. They will invest in large cattle ranches.
A. Our economy develop.
B. Our economy develops.
A. When Doris went two school, I told her that we played in the neighborhood.
B. When Doris went to school, I told her that we played in the neighborhood.
A. She have worried about the traffic jam.
B. She is worried about the traffic jam.
A. The hurricanes is huge.
B. The hurricane is huge.
A. His girlfriend likes town.
B. His girlfriend likes the town.
A. I has reported that they verify the country's nuclear programs.
B. He has reported that they verify the country's nuclear programs.
A. Girls and adults enjoy an clean and comfortable environment.
B. Girls and adults enjoy a clean and comfortable environment.
A. I keep in mind that you looks forward to that.
B. I keep in mind that you look forward to that.
A. They're support our economy since 1913!
B. They're supporting our economy since 1913!
A. You're gift to Ted is nice.
B. Your gift to Ted is nice.
A. The community is likes a helping hand.
B. The community is like a helping hand.
A. Sophronia are a four-year-old girl.
B. Sophronia is a four-year-old girl.
A. Help your neighbors too prevent violence!
B. Help your neighbors to prevent violence!
A. Her former husband keeps active by play bridge.
B. Her former husband keeps active by playing bridge.
A. Help can to benefit our community.
B. Help can benefit our community.
A. You cared enough about the lives of peoples.
B. You cared enough about the lives of people.
A. The most important investment can make a measurable difference every days!
B. The most important investment can make a measurable difference every day!
A. I sat into the rain and snow.
B. I sat in the rain and snow.
A. That money are the first gift from my mother.
B. That money is the first gift from my mother.
A. The money supports yours family.
B. The money supports your family.
A. Do you feel the positive results off their success?
B. Do you feel the positive results of their success?
```

Listing E.4: Self-made error corpus with short sentences (English)

E.2.3. Self-made Error Corpus (German)

This corpus is analog to the self-made error corpus for English. It consists of German sentences which we made up from different sentence parts out of the Universität Leipzig Wortschatz corpus [QH06] for German.

```
A. Das können der Wertpapieraufseher nicht ändern.
B. Das können die Wertpapieraufseher nicht ändern.
A. Es kam zum Streit, weil die Mieten höher seien als die Zinsen für geleiht Geld.
```

```
4  B. Es kam zum Streit, weil die Mieten höher seien als die Zinsen für geliehenes Geld.
5  A. Ich hab das aber nur gemacht, weil die zweite Gruppe der Meinung war, daß Duo habe sich
       vorgedrängelt.
6  B. Ich hab das aber nur gemacht, weil die zweite Gruppe der Meinung war, das Duo habe sich
       vorgedrängelt.
7  A. Und es ist wichtig für Demokratie wie Fortschritt, doch oft ist das Zurücktreten entscheiden
       .
8  B. Und es ist wichtig für Demokratie wie Fortschritt, doch oft ist das Zurücktreten
       entscheidend.
9  A. Der Siegermodell soll wie in den Jahren 1994/95 um über 60 Prozent jäh abstürzen.
10 B. Das Siegermodell soll wie in den Jahren 1994/95 um über 60 Prozent jäh abstürzen.
11 A. Eines Tages in Mai '99 standen sich zwei Subclans gegenüber.
12 B. Eines Tages im Mai '99 standen sich zwei Subclans gegenüber.
13 A. Ich brauchst Lothar nicht dauernd zu sagen, daß es ohne Opfer nicht mehr geht.
14 B. Ich brauche Lothar nicht dauernd zu sagen, daß es ohne Opfer nicht mehr geht.
15 A. Eine breite Zustimmung gilt als eine der zuverlässigste Methoden zur Identifizierung von
       Straftätern.
16 B. Eine breite Zustimmung gilt als eine der zuverlässigsten Methoden zur Identifizierung von
       Straftätern.
17 A. Stanczyk nannte es beunruhigend, daß die Stimmung im ORB nicht die besten ist, um die
       Leichen seiner Männer zu bergen.
18 B. Stanczyk nannte es beunruhigend, daß die Stimmung im ORB nicht die beste ist, um die Leichen
        seiner Männer zu bergen.
19 A. Wobei zu bemerken wäre, das die anderen Euro-Länder ihre Leitzinsen zurücknehmen und sich
       dem Wettbewerb stellen.
20 B. Wobei zu bemerken wäre, daß die anderen Euro-Länder ihre Leitzinsen zurücknehmen und sich
       dem Wettbewerb stellen.
21 A. Ohne ihm ist auch der Zusammenbruch der Beziehungen mit Jordanien nur noch eine Frage von
       Tagen.
22 B. Ohne ihn ist auch der Zusammenbruch der Beziehungen mit Jordanien nur noch eine Frage von
       Tagen.
23 A. Und fast 90 Prozent der Anwender gaben an, im vergangenem Monat online gegangen zu sein.
24 B. Und fast 90 Prozent der Anwender gaben an, im vergangenen Monat online gegangen zu sein.
25 A. Zu Defa-Zeit waren es nur noch rund 1,3 Milliarden Euro.
26 B. Zu Defa-Zeiten waren es nur noch rund 1,3 Milliarden Euro.
27 A. In diesem Falle könnten es für die Reiseveranstalter zu Verlusten im siebenstelligen Bereich
        kommen.
28 B. In diesem Falle könnte es für die Reiseveranstalter zu Verlusten im siebenstelligen Bereich
       kommen.
29 A. Es kann nicht darum gehen, Kinder wieder zu Lesen zu bringen.
30 B. Es kann nicht darum gehen, Kinder wieder zum Lesen zu bringen.
31 A. Natürlich deutlich stärker wie die Beitragseinahmen werden nach den Angaben des Verbands die
        Leistungen an die Versicherten zunehmen.
32 B. Natürlich deutlich stärker als die Beitragseinahmen werden nach den Angaben des Verbands die
        Leistungen an die Versicherten zunehmen.
33 A. Doch dafür ist der Aufnahmetest um so selektiver.
34 B. Doch dafür ist der Aufnahmetest umso selektiver.
35 A. Ich bin stolz auf das was im Augenblick geschieht.
36 B. Ich bin stolz auf das, was im Augenblick geschieht.
37 A. Und ich bin dankbar, daß ich diese düstere Prophetie erst las, als mir dreißig Jahre älter
       war.
38 B. Und ich bin dankbar, daß ich diese düstere Prophetie erst las, als ich dreißig Jahre älter
       war.
39 A. Ein Vierzehnjähriger kann das besser, als viele Veröffentlichungen geglaubt machen wollen.
40 B. Ein Vierzehnjähriger kann das besser, als viele Veröffentlichungen glauben machen wollen.
41 A. Nun gelten der alte Gegensatz nicht mehr, heißt es im Bundestag.
42 B. Nun gelten die alten Gegensätze nicht mehr, heißt es im Bundestag.
43 A. Daher ist es für die Welt Wirtschaft von besonderer Bedeutung, findet Boehnke.
44 B. Daher ist es für die Weltwirtschaft von besonderer Bedeutung, findet Boehnke.
45 A. Dann aber kommt es zu manchen interkulturellen Aha-Erlebnis.
46 B. Dann aber kommt es zu manchem interkulturellen Aha-Erlebnis.
47 A. Zu dem gibt es eine direkte Verbindung.
```

```
B. Zudem gibt es eine direkte Verbindung.
A. Das ganze heiß inkrementelle Suche und bedeutet, daß der Anleger stets einen Preis in der Mitte der Brief- und Geldspanne erhält.
B. Das ganze heißt inkrementelle Suche und bedeutet, daß der Anleger stets einen Preis in der Mitte der Brief- und Geldspanne erhält.
A. Der höhere Rückgang bedeutet, daß der Veranstalter ein billig Anbieter ist.
B. Der höhere Rückgang bedeutet, daß der Veranstalter ein Billiganbieter ist.
A. Der Haftbefehl wurde außer Vollzug ersetzt, weil die Zustimmung der Wohnungsbaugesellschaft fehlt.
B. Der Haftbefehl wurde außer Vollzug gesetzt, weil die Zustimmung der Wohnungsbaugesellschaft fehlt.
A. Und der Kanzler selbst sprach von eines erfreulichen Ergebnises.
B. Und der Kanzler selbst sprach von einem erfreulichen Ergebnis.
A. Ein Antwort hätte deshalb längst gegeben sein müssen, weil die Mannschaft spielerisch mehr als der Gegner zu bieten hatte und auch in sich gut harmonierte.
B. Eine Antwort hätte deshalb längst gegeben sein müssen, weil die Mannschaft spielerisch mehr als der Gegner zu bieten hatte und auch in sich gut harmonierte.
A. Sie ist eine der wenigen, die wo sich bis heute stark sozial engagiert.
B. Sie ist eine der wenigen, die sich bis heute stark sozial engagiert.
A. Mir geht es nicht um das Bruchstück eines größeren, beim Aufprall zerschellten Exemplar.
B. Mir geht es nicht um das Bruchstück eines größeren, beim Aufprall zerschellten Exemplars.
A. Er ist ein Star, nicht weil er der einzigste Mond im Sonnensystem ist.
B. Er ist ein Star, nicht weil er der einzige Mond im Sonnensystem ist.
A. Sie mußten bislang wegen den hohen Marketingkosten profitabler arbeiten als etwa Buchläden in der Stadt.
B. Sie mußten bislang wegen der hohen Marketingkosten profitabler arbeiten als etwa Buchläden in der Stadt.
A. Erst kürzlich erhielt er auf diesem Weg direkt vor dem Fernseher Waren aus einem Bestand von über 100 000 Artikeln von dem Versandhändler.
B. Erst kürzlich erhielt er auf diesem Weg direkt vor dem Fernseher Waren aus einem Bestand von über 100 000 Artikeln des Versandhändlers.
A. Interessant ist auch, daß die Betriebe diese Tarifverträge erfüllten können.
B. Interessant ist auch, daß die Betriebe diese Tarifverträge erfüllen können.
A. Daß weiß keiner besser als die Sekretärin des inzwischen verstorbenen katholischen Stadtdechanten, der zugleich Standortpfarrer war.
B. Das weiß keiner besser als die Sekretärin des inzwischen verstorbenen katholischen Stadtdechanten, der zugleich Standortpfarrer war.
A. Er war ein Friseur, und muß sich damit nicht abfinden.
B. Er war ein Friseur und muß sich damit nicht abfinden.
A. Es sei allerdings nicht nötig, so etwas fordern.
B. Es sei allerdings nicht nötig, so etwas zu fordern.
A. Vielleicht hat er sich deshalb mit den falschen Dokumenten ein geschlichen, so die CDU-Landtagsfraktion.
B. Vielleicht hat er sich deshalb mit den falschen Dokumenten eingeschlichen, so die CDU-Landtagsfraktion.
A. Zirkus kann es auch dann geben, wann die Länderchefs am heutigen Donnerstag mal wieder über Gebühren und Strukturen bei ARD und ZDF und über den Fall Sarkuhi reden.
B. Zirkus kann es auch dann geben, wenn die Länderchefs am heutigen Donnerstag mal wieder über Gebühren und Strukturen bei ARD und ZDF und über den Fall Sarkuhi reden.
A. Doch in der Nacht vom 25. zum 26. April 1986 um 1 Uhr 23 Minuten und 40 Sekunden, ging ein Stück der Welt verloren, als der Regen kam und mit ihm der Notstand.
B. Noch in der Nacht vom 25. zum 26. April 1986 um 1 Uhr 23 Minuten und 40 Sekunden, ging ein Stück der Welt verloren, als der Regen kam und mit ihm der Notstand.
A. Viele Funktionen des Claustrums ist noch sehr weit von einer ausgeprägten kommerziellen Orientierung entfernt.
B. Viele Funktionen des Claustrums sind noch sehr weit von einer ausgeprägten kommerziellen Orientierung entfernt.
A. Nur plötzlich präsentieren sich die Führer der kleinen Parteien.
B. Nun plötzlich präsentieren sich die Führer der kleinen Parteien.
A. Trotz dem müssen sie nicht unbedingt in warme, südliche Länder fliegen.
B. Trotzdem müssen sie nicht unbedingt in warme, südliche Länder fliegen.
A. Leider Wissen die meisten gar nicht, was draußen auf der Straße wirklich abgeht.
```

90 B. Leider wissen die meisten gar nicht, was draußen auf der Straße wirklich abgeht.
91 A. In Juli und August sollen dann auch die im dritten Bauabschnitt geplanten Gewerberäume fertig sein, insgesamt will das Eigentümerkonsortium 300 Millionen Mark in das Viktoria Quartier investieren.
92 B. Im Juli und August sollen dann auch die im dritten Bauabschnitt geplanten Gewerberäume fertig sein, insgesamt will das Eigentümerkonsortium 300 Millionen Mark in das Viktoria Quartier investieren.
93 A. Ich hoffe, daß noch nicht abgeschlossen werde.
94 B. Ich hoffe, daß noch nicht abgeschlossen wurde.
95 A. Das Bild ändern sich erst, wenn die Streu auf dem Waldboden verbrannt ist.
96 B. Das Bild ändert sich erst, wenn die Streu auf dem Waldboden verbrannt ist.
97 A. Der Radler wurde bei dem Zusammenprall mit dem Autos überrollt.
98 B. Der Radler wurde bei dem Zusammenprall mit dem Auto überrollt.
99 A. Bei einem Volksfest am vergangenen Wochenende in München war Anja Osthoff allerdings beleidigend worden.
100 B. Bei einem Volksfest am vergangenen Wochenende in München war Anja Osthoff allerdings beleidigt worden.
101 A. Aber auch die Öffentlichkeit sei nicht ausreichend über ihr Rechte informiert.
102 B. Aber auch die Öffentlichkeit sei nicht ausreichend über ihre Rechte informiert.
103 A. Gleichwohl gab es am vergangenen Wochenende wahre schlachten.
104 B. Gleichwohl gab es am vergangenen Wochenende wahre Schlachten.
105 A. Diese Forderung macht sich außerdem gut in dieser Zeiten, in den kommenden Tagen soll demnach eine Entscheidung fallen.
106 B. Diese Forderung macht sich außerdem gut in diesen Zeiten, in den kommenden Tagen soll demnach eine Entscheidung fallen.
107 A. Baxter, der die Einnahme von Metamphetaminen überführt worden war, sagte, die Entscheidung sei im Unternehmen nicht bekannt.
108 B. Baxter, der der Einnahme von Metamphetaminen überführt worden war, sagte, die Entscheidung sei im Unternehmen nicht bekannt.
109 A. Ich war geschockt frustriert zugleich, weil die klassische Sozialplanlösung nicht mehr funktioniert.
110 B. Ich war geschockt und frustriert zugleich, weil die klassische Sozialplanlösung nicht mehr funktioniert.
111 A. Die jetzt Situation schafft zusätzliches Druckpotential, weil die Sehne zuvor klinisch gut aussah.
112 B. Die jetzige Situation schafft zusätzliches Druckpotential, weil die Sehne zuvor klinisch gut aussah.
113 A. Diese hatte die Fascination im Jahr 2003 ins Leben gerufen, weil die Bestimmungen keine und mittlere Unternehmungen vor den Auswirkungen eines solchen Preiskampfs schützen sollten.
114 B. Diese hatte die Fascination im Jahr 2003 ins Leben gerufen, weil die Bestimmungen kleine und mittlere Unternehmungen vor den Auswirkungen eines solchen Preiskampfs schützen sollten.
115 A. Das Unternehmen sieht sich inzwischen auf dem deutschen Markt im Wettbewerb mit jedes Direktanbieters.
116 B. Das Unternehmen sieht sich inzwischen auf dem deutschen Markt im Wettbewerb mit jedem Direktanbieter.
117 A. Die Union plant im Fall eines Sieg bei der Bundestagswahl 2002 die notwendigen Konsequenzen zu ziehen.
118 B. Die Union plant im Fall eines Sieges bei der Bundestagswahl 2002 die notwendigen Konsequenzen zu ziehen.
119 A. Privatleute haben sein Häuser zur Verfügung gestellt, sich aber das Recht der Eigennutzung von vier Wochen pro Jahr vorbehalten.
120 B. Privatleute haben ihre Häuser zur Verfügung gestellt, sich aber das Recht der Eigennutzung von vier Wochen pro Jahr vorbehalten.
121 A. Außerdem sei ihnen das Original-Fotomaterial nicht zur Verfügung stellt geworden, teilte Finanzsenator Elmar Pieroth (CDU) mit.
122 B. Außerdem sei ihnen das Original-Fotomaterial nicht zur Verfügung gestellt worden, teilte Finanzsenator Elmar Pieroth (CDU) mit.
123 A. Der in Berlin lebende Mann hatte seine Wohnung langfristig an ein Hotelgesellschaft zur Vermietung zur Verfügung gestellt, und über Progamme und Flyer wurde dieses schon werbewirksam in Umlauf gebracht.
124 B. Der in Berlin lebende Mann hatte seine Wohnung langfristig an eine Hotelgesellschaft zur Vermietung zur Verfügung gestellt, und über Progamme und Flyer wurde dieses schon

```
      werbewirksam in Umlauf gebracht.
125 A. Fraktionen dürften die ihnen vom Staat zur Verfügungen gestellten Garantien in Höhe von 85
      Prozent der Gesamtkosten beantragen.
126 B. Fraktionen dürften die ihnen vom Staat zur Verfügung gestellten Garantien in Höhe von 85
      Prozent der Gesamtkosten beantragen.
127 A. Das CSU-Politiker verlangt weitere personelle Konsequenzen in den russischen Behörden.
128 B. Der CSU-Politiker verlangt weitere personelle Konsequenzen in den russischen Behörden.
129 A. Die Entscheidung des Deutschen Börse AG ist geplatzt.
130 B. Die Entscheidung der Deutschen Börse AG ist geplatzt.
131 A. So wäre es der vierte Chief Executive Officer von Coca-Cola Deutschland, den größtem
      Mehrwegabfüller in der Bundesrepublik.
132 B. So wäre es der vierte Chief Executive Officer von Coca-Cola Deutschland, dem größten
      Mehrwegabfüller in der Bundesrepublik.
133 A. Hochtief sah dich gezwungen, die Commerzbank einzuschalten, weil die Zinsen dann etwas höher
       sind - schon sitzen sie in der Falle.
134 B. Hochtief sah sich gezwungen, die Commerzbank einzuschalten, weil die Zinsen dann etwas höher
       sind - schon sitzen sie in der Falle.
135 A. Viel leicht verlieren Sie so oder so, weil die Hunde das Gewicht sonst nicht schaffen.
136 B. Vielleicht verlieren Sie so oder so, weil die Hunde das Gewicht sonst nicht schaffen.
137 A. Doch angenommen, sie verlängern ihr Festgeld jetzt über das Jahresende hinaus, weil die
      klassischen Sozialplanlösung nicht mehr funktioniert.
138 B. Doch angenommen, sie verlängern ihr Festgeld jetzt über das Jahresende hinaus, weil die
      klassische Sozialplanlösung nicht mehr funktioniert.
139 A. Im Januar 1998 schossen Polizisten die Schulen, weil die Mehrkosten gegenüber den gängigen
      Risikomodellen 15 bis 20 Prozent betragen.
140 B. Im Januar 1998 schlossen Polizisten die Schulen, weil die Mehrkosten gegenüber den gängigen
      Risikomodellen 15 bis 20 Prozent betragen.
141 A. Der Terrier fliegen durch die Luft und markiert einen Punkt.
142 B. Der Terrier fliegt durch die Luft und markiert einen Punkt.
143 A. Wir haben schon seit Jahren keinen guten Nachwuchsspielern mehr herausgebracht.
144 B. Wir haben schon seit Jahren keinen guten Nachwuchsspieler mehr herausgebracht.
145 A. Diese Quote liege auch deutlich höher als das Brutto Einkommen auf jedem dieser
      Arbeitsplätze.
146 B. Diese Quote liege auch deutlich höher als das Bruttoeinkommen auf jedem dieser Arbeitsplätze
      .
147 A. Geht es steil Berg auf, sollte er mitlaufen, weil die weißen Mauern sein Bild tragen.
148 B. Geht es steil bergauf, sollte er mitlaufen, weil die weißen Mauern sein Bild tragen.
149 A. Bloß gebaut wurde kein, weil die "Lokomotive" Bauwirtschaft weggefallen ist.
150 B. Bloß gebaut wurde keiner, weil die "Lokomotive" Bauwirtschaft weggefallen ist.
151 A. Trotzdem bleibt das englische Regierungssprache, weil die Bevölkerung vor ihm noch nie einen
       Schwarzen gesehen hatte.
152 B. Trotzdem bleibt das Englische Regierungssprache, weil die Bevölkerung vor ihm noch nie einen
       Schwarzen gesehen hatte.
153 A. Die Arbeitslosigkeit ist deswegen in dem letzten Monaten so in die Höhe geschnellt, weil die
       Ausbilder ehrenamtlich arbeiten.
154 B. Die Arbeitslosigkeit ist deswegen in den letzten Monaten so in die Höhe geschnellt, weil die
       Ausbilder ehrenamtlich arbeiten.
155 A. Die Praxisstunden ist bei den meisten Vereinen kostenlos, weil die Bauarbeiter ihren Job
      verflixt ernst nehmen.
156 B. Die Praxisstunden sind bei den meisten Vereinen kostenlos, weil die Bauarbeiter ihren Job
      verflixt ernst nehmen.
157 A. Dieser Betrag sei damit deutlich höher als in den neuen Bundesländern ins gesamt.
158 B. Dieser Betrag sei damit deutlich höher als in den neuen Bundesländern insgesamt.
159 A. Ich habe auf eine Wunder gehofft - obwohl es die Anklage durch Den Haag gibt.
160 B. Ich habe auf ein Wunder gehofft - obwohl es die Anklage durch Den Haag gibt.
161 A. Die 7:0-Gala gegen Lübeck ist noch gar nicht solange her, da haben wir unser Pulver total
      verschossen.
162 B. Die 7:0-Gala gegen Lübeck ist noch gar nicht so lange her, da haben wir unser Pulver total
      verschossen.
163 A. Es werden Verhandlungen mit Milosevic statt finden, obwohl es alle gleichermaßen angehen
      müßte.
```

164 B. Es werden Verhandlungen mit Milosevic stattfinden, obwohl es alle gleichermaßen angehen müßte.
165 A. Am Ende war Sieg Ellerbeks verdient, weil die Wochendaten zu den US-Öl- und Benzinvorräten besser ausfielen als erwartet.
166 B. Am Ende war der Sieg Ellerbeks verdient, weil die Wochendaten zu den US-Öl- und Benzinvorräten besser ausfielen als erwartet.
167 A. Mittwochabend geriet er er besonders stark unter Druck, weil die Haut am Körper zu sehr spannt.
168 B. Mittwochabend geriet er besonders stark unter Druck, weil die Haut am Körper zu sehr spannt.
169 A. Den noch könnte auch dies nach dem Gesetz gegen Wettbewerbsbeschränkungen verboten sein, weil die Unternehmen hier Rückenwind aus der Weltwirtschaft spüren und der globale Handel boomt.
170 B. Dennoch könnte auch dies nach dem Gesetz gegen Wettbewerbsbeschränkungen verboten sein, weil die Unternehmen hier Rückenwind aus der Weltwirtschaft spüren und der globale Handel boomt.
171 A. Die Behörde sind zumeist dankbar dafür, weil die Raketen als sehr zuverlässig gelten.
172 B. Die Behörden sind zumeist dankbar dafür, weil die Raketen als sehr zuverlässig gelten.
173 A. Vielleicht auch, weil zu die Kantine gleich macht.
174 B. Vielleicht auch, weil die Kantine gleich zumacht.
175 A. Hamburg schneidet vergleichsweise gut, weil die Essener der bedeutendste Abnehmer von norwegischem Erdgas sind.
176 B. Hamburg schneidet vergleichsweise gut ab, weil die Essener der bedeutendste Abnehmer von norwegischem Erdgas sind.
177 A. Das hatte die ETA in einem Schreiben an die die Zeitung Gara bestätigt, die darüber am kommenden Dienstag beraten wird.
178 B. Das hatte die ETA in einem Schreiben an die Zeitung Gara bestätigt, die darüber am kommenden Dienstag beraten wird.
179 A. Auswirkungen auf den Verfassungsprozeß innerhalb der Europäischen Union haben die Türkei neue Hoffnungen auf einen EU-Beitritt gemacht.
180 B. Auswirkungen auf den Verfassungsprozeß innerhalb der Europäischen Union haben der Türkei neue Hoffnungen auf einen EU-Beitritt gemacht.
181 A. Erste Entwürfe für das neue Angebot werden im Unternehmen seid einiger Zeit verfolgt.
182 B. Erste Entwürfe für das neue Angebot werden im Unternehmen seit einiger Zeit verfolgt.
183 A. Microsoft und die ebenfalls klagende US-Regierung hatte sich am Freitag jedoch so geirrt wie selten.
184 B. Microsoft und die ebenfalls klagende US-Regierung hatten sich am Freitag jedoch so geirrt wie selten.
185 A. Die meisten Meteorologen hatten sich am freitags auf einen Vergleich verständigt.
186 B. Die meisten Meteorologen hatten sich am Freitag auf einen Vergleich verständigt.
187 A. Die Tiere halten Sie auseinander, weil die Nachgeburt als hochinfektiös gilt.
188 B. Die Tiere halten sie auseinander, weil die Nachgeburt als hochinfektiös gilt.
189 A. Ich habe jedenfalls vor, noch in diesem Jahr, ein Zeichen zu setzen, daß die Bahn nichts nur Hilfe fordert, sondern sich auch selber hilft.
190 B. Ich habe jedenfalls vor, noch in diesem Jahr, ein Zeichen zu setzen, daß die Bahn nicht nur Hilfe fordert, sondern sich auch selber hilft.
191 A. Als ich abends nach Hause kam, hatte sie ein ganz seltsames gefühlt.
192 B. Als ich abends nach Hause kam, hatte sie ein ganz seltsames Gefühl.
193 A. Das komme das Land auf Dauer teuer stehen, weil die einfach sehr interessante Sachen zu erzählen haben.
194 B. Das komme das Land auf Dauer teuer zu stehen, weil die einfach sehr interessante Sachen zu erzählen haben.
195 A. Es ist noch gar richtig zu Ende, er macht nur eine Pause.
196 B. Es ist noch gar nicht richtig zu Ende, er macht nur eine Pause.
197 A. Rund 30 Künstlerinnen und Künstler sind in in der Ausstellung dokumentiert.
198 B. Rund 30 Künstlerinnen und Künstler sind in der Ausstellung dokumentiert.
199 A. Auch Großbritannien oder die Niederlande besitzt einen Vorrat von Impfstoff, in den USA stieg das Volumen gar um 150 Prozent.
200 B. Auch Großbritannien oder die Niederlande besitzen einen Vorrat von Impfstoff, in den USA stieg das Volumen gar um 150 Prozent.
201 A. Das Jahr 1994 stufte Kopper als das drittbeste im Geschichte der deutschen Psychiatrie dar.
202 B. Das Jahr 1994 stufte Kopper als das drittbeste in der Geschichte der deutschen Psychiatrie dar.

```
A. Heute stehen "Katzen, Mäuse und Ratten - Tierchen in Erwachsenencomics" auf dem Programm, sagte Schulz sein Manager Sauerland, der in Leipzig vorsichtig von "einer weitere Vorwärtsentwicklung" sprach.
B. Heute stehen "Katzen, Mäuse und Ratten - Tierchen in Erwachsenencomics" auf dem Programm, sagte Schulz' Manager Sauerland, der in Leipzig vorsichtig von "einer weitere Vorwärtsentwicklung" sprach.
A. Luftangriffe und Landung von Fallschirmjägern in "feindlichem Gebiet" stehen auf Programm, das man in der Präsidentenadministration heute noch nicht ändern wollte.
B. Luftangriffe und Landung von Fallschirmjägern in "feindlichem Gebiet" stehen auf dem Programm, das man in der Präsidentenadministration heute noch nicht ändern wollte.
A. Die Polizei verhinderte das Anbringen der Tafel, weil die Stadt bereits in 1997 mit dem radikalen Sparen begonnen hat.
B. Die Polizei verhinderte das Anbringen der Tafel, weil die Stadt bereits 1997 mit dem radikalen Sparen begonnen hat.
A. Möglich wurde dies alles nur, weil die Spieler eleganter sind, mehr kombinieren und er viel offensiver zugeht.
B. Möglich wurde dies alles nur, weil die Spieler eleganter sind, mehr kombinieren und es viel offensiver zugeht.
A. Es macht mehr Spaß, weil die Kirche weit kleiner wie der Dom sei.
B. Es macht mehr Spaß, weil die Kirche weit kleiner als der Dom sei.
A. Der Krach droht Zeit weise sogar nachts, weil die Auswertung der Quellen noch nicht abgeschlossen wurde.
B. Der Krach droht zeitweise sogar nachts, weil die Auswertung der Quellen noch nicht abgeschlossen wurde.
A. Es ist wahr, daß wir wie Marionetten an ihren Fäden zu zappeln.
B. Es ist wahr, daß wir wie Marionetten an ihren Fäden zappeln.
A. Doch nun, weil die Eltern eine Unterforderung ihrer Kinder befürchten.
B. Doch nur, weil die Eltern eine Unterforderung ihrer Kinder befürchten.
A. Man haben nicht von vornherein dort gespielt, weil die Produkte teuer sind.
B. Man habe nicht von vornherein dort gespielt, weil die Produkte teuer sind.
A. Ein Kampf um die Macht, das mehr als einmal überrascht.
B. Ein Kampf um die Macht, der mehr als einmal überrascht.
A. Möglich war dies nur, weil die Baufirma insolvent ißt.
B. Möglich war dies nur, weil die Baufirma insolvent ist.
A. Aber auch lobt Renate und Andreas, weil die Bettenkapazität um 15,9 Prozent gestiegen ist.
B. Aber auch sie lobt Renate und Andreas, weil die Bettenkapazität um 15,9 Prozent gestiegen ist.
A. Er spricht sogar von einem "Glücksfall", weil die Glashütte seine technischen Anlagen umbaut.
B. Er spricht sogar von einem "Glücksfall", weil die Glashütte ihre technischen Anlagen umbaut.
A. Dieser sei vermeidbar gewesen weil die Bezirke solche Statistiken nicht führen.
B. Dieser sei vermeidbar gewesen, weil die Bezirke solche Statistiken nicht führen.
A. Das Angebot reicht vom Holzbalken und Ziegeln über Türen, Fenster, Parkett und Treppen bis hin zu "Fetischismus und Moderne" bei den Kulturwissenschaftlern.
B. Das Angebot reicht von Holzbalken und Ziegeln über Türen, Fenster, Parkett und Treppen bis hin zu "Fetischismus und Moderne" bei den Kulturwissenschaftlern.
A. Beide haben massive finanzielle Schwierigkeiten und beide wissen nicht, wie wie es in der nächsten Saison aussieht, darüber will Manager Peter John Lee erst kommende Woche reden.
B. Beide haben massive finanzielle Schwierigkeiten und beide wissen nicht, wie es in der nächsten Saison aussieht, darüber will Manager Peter John Lee erst kommende Woche reden.
A. Was nicht bedeutet, daß man keine "Suchen"-Button mehr anklicken muß.
B. Was nicht bedeutet, daß man keinen "Suchen"-Button mehr anklicken muß.
A. Kleinere Zwischenfall melden sie gar nicht erst, weil die Landesbank haftet.
B. Kleinere Zwischenfälle melden sie gar nicht erst, weil die Landesbank haftet.
A. Es geht uns auch Nahe, weil die Mieter damit einverstanden sein müssen.
B. Es geht uns auch nahe, weil die Mieter damit einverstanden sein müssen.
A. So müssen Muttertiere im Stall ablammen, weil die Übernahme nicht zu Stande gekommen ist.
B. So müssen Muttertiere im Stall ablammen, weil die Übernahme nicht zustande gekommen ist.
A. ITN stellte ihm aber in den Mittelpunkt seiner Klage, weil die Unsicherheiten verschwunden seien.
B. ITN stellte ihn aber in den Mittelpunkt seiner Klage, weil die Unsicherheiten verschwunden seien.
```

```
A. Ruhrgas könnte da bei Priorität erhalten, weil die Grünen es geschluckt haben.
B. Ruhrgas könnte dabei Priorität erhalten, weil die Grünen es geschluckt haben.
A. Bis zu 26. Februar muß die Stadt ihre Stellungnahme abgeben, weil die Rinder doch
     verschieden gezeichnet sind und einen unterschiedlichen Körperbau haben.
B. Bis zum 26. Februar muß die Stadt ihre Stellungnahme abgeben, weil die Rinder doch
     verschieden gezeichnet sind und einen unterschiedlichen Körperbau haben.
A. Er kann nicht joggen, weil die Ball Sport Halle Frankfurt-Höchst am ursprünglichen Termin
     11. Februar belegt ist.
B. Er kann nicht joggen, weil die Ballsporthalle Frankfurt-Höchst am ursprünglichen Termin 11.
     Februar belegt ist.
A. Das warst vor elf Jahren, als sie sich zwar lachend verbat, nicht als Frau Merkel tituliert
     zu werden, aber dann nicht auf dem Gast herumhackte.
B. Das war vor elf Jahren, als sie sich zwar lachend verbat, nicht als Frau Merkel tituliert zu
      werden, aber dann nicht auf dem Gast herumhackte.
A. Das brachte zwar ein Versäumnisurteil, aber aber keine Anstellung.
B. Das brachte zwar ein Versäumnisurteil, aber keine Anstellung.
A. Bei der Suche nach eine Lehrstelle sollte man sich nicht trennen.
B. Bei der Suche nach einer Lehrstelle sollte man sich nicht trennen.
A. Dann dürfen man sich aber nicht auf Unternehmen beschränken, sondern muß sich immer auf den
     Etatentwurf stützen, den Berthold Depper schon Ende September vorgelegt hatte.
B. Dann darf man sich aber nicht auf Unternehmen beschränken, sondern muß sich immer auf den
     Etatentwurf stützen, den Berthold Depper schon Ende September vorgelegt hatte.
A. Weitere Gespräche über ein das Zusammengehen mit ThyssenKrupp will man, das ist die
     schlechte Nachricht, beibehalten.
B. Weitere Gespräche über ein Zusammengehen mit ThyssenKrupp will man, das ist die schlechte
     Nachricht, beibehalten.
```

Listing E.5: Self-made error corpus (German)