Thesis for The Degree of Doctor of Philosophy

Learning Language (with) Grammars

From Teaching Latin to Learning Domain-Specific Grammars

Herbert Lange

UNIVERSITY OF GOTHENBURG

Division of Functional Programming
Department of Computer Science & Engineering
University of Gothenburg
Gothenburg, Sweden, 2020

Learning Language (with) Grammars:
From Teaching Latin to Learning Domain-Specific Grammars

Herbert Lange

ISBN:
978-91-7833-986-0 (Print)
978-91-7833-987-7 (PDF)
http://hdl.handle.net/2077/65453

Technical Report No 185D
Department of Computer Science & Engineering
Division of Functional Programming
University of Gothenburg
Gothenburg, Sweden

Cover image:
Pottery Fan: Douris Man with wax tablet,
modified by Herbert Lange and Catharina Jerkbrant
Source: Wikimedia
https://de.wikipedia.org/wiki/Datei:Douris_Man_with_wax_tablet.jpg
License: CC-BY-SA 3.0
https://creativecommons.org/licenses/by-sa/3.0

Author photo:
Anneli Andersson

This thesis has been prepared using LuaLaTeX.
Printed by Chalmers digitaltryck,
Gothenburg, Sweden 2020.

— *Introibo ad altare Dei.*
Halted, he peered down the dark winding stairs
and called out coarsely:
— Come up, Kinch! Come up, you fearful jesuit!

James Joyce, Ulysses
Telemachus

Abstract

This thesis describes work in three areas: grammar engineering, computer-assisted language learning and grammar learning. These three parts are connected by the concept of a grammar-based language learning application. Two types of grammars are of concern. The first we call resource grammars, extensive descriptions a natural languages. Part I focuses on this kind of grammars. The other are domain-specific or application-specific grammars. These grammars only describe a fragment of natural language that is determined by the domain of a certain application. Domain-specific grammars are relevant for Part II and Part III. Another important distinction is between humans learning a new natural language using computational grammars (Part II) and computers learning grammars from example sentences (Part III).

Part I of this thesis focuses on grammar engineering and grammar testing. It describes the development and evaluation of a computational resource grammar for Latin. Latin is known for its rich morphology and free word order, both have to be handled in a computationally efficient way. A special focus is on methods how computational grammars can be evaluated using corpus data. Such an evaluation is presented for the Latin resource grammar.

Part II, the central part, describes a computer-assisted language learning application based on domain-specific grammars. The language learning application demonstrates how computational grammars can be used to guide the user input and how language learning exercises can be modeled as grammars. This allows us to put computational grammars in the center of the design of language learning exercises used to help humans learn new languages.

Part III, the final part, is dedicated to a method to learn domain- or application-specific grammars based on a wide-coverage grammar and small sets of example sentences. Here a computer is learning a grammar for a fragment of a natural language from example sentences, potentially without any additional human intervention. These learned grammars can be based e.g. on the Latin resource grammar described in Part II and used as domain-specific lesson grammars in the language learning application described Part II.

Keywords: Latin, Latin syntax, Latin morphology, Grammar engineering, Grammar testing, Corpus-based evaluation, Computer-Assisted Language Learning, Grammar learning, Constraint-satisfaction, Constraint-optimization

Acknowledgment

So many people have helped me on my way to finish my PhD, not all can be mentioned here by name and I am sure I will forget someone. If you, dear reader, feel that your name is missing from this list, be assured that it is not due to malice.

I have to thank my PhD supervisor Peter Ljunglöf. He allowed me to be as independent as I wanted to be, but still was always there with a helping hand, when I needed it. I also want to thank my co-supervisor Koen Claessen, who in the end managed to pull me over to the dark side and use constraint solvers. Finally, I am very grateful for my examiner Aarne Ranta, who also helped me a lot in my research with his incredible knowledge about grammars, languages and and everything else.

I also want to thank my opponent Johanna Björklund and my committee: Torsten Zesch, Gunhild Vidén, Elena Volodina and Richard Johannson.

Next, I want to thank my family: my parents, my brother and my aunt, who supported the crazy idea to move to Sweden to pursue my PhD studies.

Finally I have to thank a long list of friends and colleagues, specifically: Aljoscha (a great person to hang out with and play German card games), Arianna (who kept me company during a pandemic by cycling around Delsön and shares an interest in computer science and languages), David (who taught me the important skills of knitting and crocheting), Inari (with whom I shared an office, a boat, the love for languages and grammars and from time to time a sauna), Klondike (who dragged me into Gothenburg hackerspace and started his own PhD despite all my efforts), Linnea (without whom I probably would have given up on learning Swedish and who helped me to get used to Swedish culture), Pavol (who helped me to stay in touch with the humanities), Prasanth (who used to "live" in our office), Thomas (who also enjoys playing cards, a fellow ham amateur, and even owner of a sailing boat), and Víctor (with whom I spent most of my lunches and who was always interested in joining for various activities).

I am also in debt to all the people I had interesting conversations with, either over lunch or at an afterworks, my colleagues in the CSE PhD Council and at GUDK, all the people at CLASP and Språkbanken, people who shared my interests and hobbies, e.g. my fellow fencers who like to beat me up or sometimes even get beaten, and the choir that accepted my singing skills.

The MUSTE project is funded by the Swedish Research Council (Vetenskapsrådet) under grant number 621-2014-4788.

List of Publications

This thesis is based on the following publications:

[A] Herbert Lange: "Implementation of a Latin Grammar in Grammatical Framework", Published in *Proceedings of the 2nd International Conference on Digital Access to Textual Cultural Heritage (DATeCH2017), Göttingen, Germany, 2017*

[B] Herbert Lange: "An Open-Source Computational Latin Grammar: Overview and Evaluation", Submitted to *Proceedings of the 20th International Colloquium on Latin Linguistics (ICLL 2019), Las Palmas de Gran Canaria, 2019*

[C] Herbert Lange and Peter Ljunglöf: "MULLE: A Grammar-Based Latin Language Learning Tool to Supplement the Classroom Setting", Published in *Proceedings of the 5th Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA '18), Melbourn. Australia, 2018*

[D] Herbert Lange and Peter Ljunglöf: "Putting Control into Language Learning", Published in *Proceedings of the 6th International Workshop on Controlled Natural Languages (CNL 2018), Maynooth, Ireland, 2018*

[E] Herbert Lange and Peter Ljunglöf: "Learning Domain-Specific Grammars From a Small Number of Examples", Submitted to *Special Issue: Natural Language Processing in Artificial Intelligence - NLPinAI 2020, in: Series "Studies in Computational Intelligence" (SCI), Springer*

Not included are:

- Herbert Lange and Peter Ljunglöf "Demonstrating the MUSTE Language Learning Environment" Published in *Proceedings of the 7th Workshop on NLP for Computer Assisted Language Learning (NLP4CALL 2018), Stockholm, 2018*
- Herbert Lange and Peter Ljunglöf "Learning Domain-Specific Grammars From a Small Number of Examples" Published in *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART 2020) – Volume 1: NLPinAI, Valetta, Malta, 2020*

Contributions

[A] This paper (Chapter 5) is the first description of the Latin resource grammar which has been developed sinced 2013 by the author of the thesis. It presents language learning as one of the possible applications of computational grammars.

[B] This paper (Chapter 6) extends the description of the Latin resource grammar to include extensions that have been added by the author since the previous publication. A major new contribution is the description of evaluation methods for resource grammars in general and applied to the Latin grammar.

[C] This paper (Chapter 7) presents an application for computer-assisted language learning using compuational grammars. These grammars are used both for guiding the user input and for defining the learning output. The implementation is based on previous work by Peter Ljunglöf but has been reimplemented by the author of this thesis. Furthermore, the author of this thesis contributed the focus on Latin language learning, ideas on how to model lessons using grammars and the addition of gamification.

[D] This paper (Chapter 8) extends the idea of grammar-based CALL and places it in the context of controlled natural languages. The author of this thesis contributes the focus on properties of lesson grammars, including a classification within the hierarchy of controlled natural languages using the PENS classification system.

[E] This paper (Chapter 9) adds a new perspective of grammar inference and grammar learning. It focuses on a method to extracting a domain-specific subgrammar from a resource grammar using example sentences. The method has been implemented by the author of the thesis and he designed experiments for evaluation. The basic method is extended in two ways and for both extensions the effect on learnability of phenomena is presented.

Contents

List of Acronyms

ACE Attempto Controlled English. 40, 108

API application programming interface. 99, 129

CALL computer-assisted language learning. 3–5, 7, 9, 10, 15, 17, 18, 21, 33, 35, 40, 41, 48, 55–57, 99, 106, 107, 116, 122, 124, 154

CCG combinatory categorial grammar. 8

CFG context-free grammar. 15–17, 64, 125, 126

CNL controlled natural language. 4, 21, 40, 55, 57, 106–108, 115–117

COP constraint optimization problem. 18, 19, 46, 50, 129, 142, 146, 153

CSP constraint satisfaction problem. 18, 44, 46, 49, 50, 122, 129, 131, 132, 146, 153

DOP data-oriented parsing. 16, 17, 125, 126, 146

EM Expectation Maximazition. 16, 126

GF Grammatical Framework. 4, 5, 10–12, 14, 15, 17, 22–25, 27, 29–32, 38–40, 44, 48, 49, 56, 64, 65, 67, 68, 77–80, 83, 84, 86–91, 94, 99, 100, 108–110, 114, 123, 126–129, 137–139, 144, 145

GPSG generalized phrase structure grammar. 12, 72

HPSG head-driven phrase structure grammar. 12, 123, 126

ILP integer linear programming. 18, 44, 130, 135, 136, 153

LAS labeled attachment score. 138

LFG lexical-functional grammar. 12, 123

LTAG lexicalized tree-adjoining grammar. 123

MULLE MUSTE language learning environment. 98–101, 107

NLP natural language processing. 3–5, 7, 8, 19, 21, 31, 56, 62, 63, 123

PCFG probabilistic context-free grammar. 16, 17, 125, 126

PMCFG parallel multiple context-free grammar. 64

POS part of speech. 23, 29, 31, 32, 89, 90

RGL resource grammar library. 5, 15, 17, 24, 26, 29, 31, 48, 65, 77, 79, 84, 86, 89, 91, 92, 99–101, 110, 123, 129, 137–139, 144–146, 150, 151

SAT Boolean satisfiability. 17–20, 126, 153, 154

UD Universal Dependencies. 5, 8, 9, 29–32, 78, 86–90, 92, 94

Part I

Introduction and Overview

Chapter 1

Introduction

The book you are holding in your hands is the result of five years of my PhD research. And this chapter in particular is supposed to summarize the work done and the results achieved.

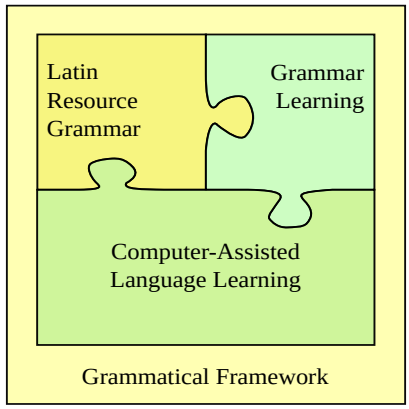


Figure 1.1: The areas relevant to this thesis

The research included in this thesis touches on three different areas and are held together by one application (Figure 1.1). The areas are natural language processing (NLP) approaches to Latin by means of grammar engineering, computer-assisted language learning (CALL) and grammar learning. This research led to the development of a computer-assisted language learning application for Latin.

These areas are already well established, but I was able to both combine them and contribute to them in a new and meaningful way. The results of these efforts are collected in this thesis. Our most obvious contribution is the creation of an intelligent CALL application for Latin, filling a gap that existed for many years. The main focus for Latin has been on traditional learning methods such as memorizing vocabulary and word inflection. But other absorbing applications suitable for practicing more complex aspects of the

language have been absent until *Duolingo* released its Latin course in August 2019.

One more aspect that surrounds, connects and permeates everything, is the use of one particular grammar formalism for computational formal grammars. We make use of the Grammatical Framework (GF) in all parts of our work. We are very fond of this grammar formalism developed in Gothenburg. GF is well suited for all our needs, but it can be replaced by other formalisms as well.

1.1 Research Questions and Contributions

Each of the three areas leads to research questions I investigated in my PhD and present the research results in this thesis. The main research questions, one for each area, are:

Q1 – Latin NLP and grammar development How can a computational grammar, especially for Latin, be implement and how can the quality of such a grammar be evaluated?

Q2 – Computer-assisted language learning How can grammar-based and multilingual controlled natural languages be used in the context of computer-assisted language learning?

Q3 – Subgrammar extraction and grammar learning Is there a (semi-) automatic way of generating domain- and application specific grammars?

In the following paragraphs I will summarize the major contributions based on these research questions.

My work on Latin NLP and grammar development involved the creation of a computational grammar for the Latin language, that provides an extensive description of its linguistic features. The grammar is described in detail in the paper in Chapter 5. The grammar has a sufficient extent to be useful in practical applications, such as our language learning application. A broader contribution that goes beyond just a Latin grammar, is the work on testing and evaluating the grammar, which is a major part of the paper in Chapter 6.

In the area of CALL we explored the use of computational grammars in language learning. We build on previous research on grammar-based text editing (Ljunglöf, 2011) and explore how this intuitive way of text input can be combined with specific grammars modeling the learning objectives. The first paper in this area (Chapter 7) describes the architecture and features of our application in general. It also describes how common CALL features such as gamification can be included to create a more immersive learning experience. The other CALL paper in Chapter 8 focuses more on the structure and properties of grammars suitable for our CALL application. The major contribution is the exploration of how controlled natural languages (CNLs) and computational grammars can be used as a central point in CALL applications, controlling the user input, allowing the automatic generation of translation exercises and modeling the learning objectives for language learning lessons.

The final part of the thesis contains contributions to general grammar learning from examples. Unlike related approaches such as grammar inference and data-oriented parsing, we created a method to learn a subgrammar from a larger resource grammar. It is described in Chapter 9 together with two extensions. We cannot only learn a suitable subgrammar from a set of positive examples but also from negative examples. And we can merge grammar rules to get an even more domain- and application-specific grammar.

1.2 Main Results

The main result of this work is the creation of the three jigsaw pieces seen in Figure 1.1. We created and implemented three different components: a computational Latin grammar, a grammar-based CALL application and a method to learn domain-specific grammars from example sentences.

The computational Latin grammar is released as free and open source and included in the Grammatical Framework resource grammar library. It can be used as a linguistic resource for building NLP applications, such as our CALL application. Moreover, I evaluated the coverage of the Latin grammar on a corpus taken from the Universal Dependencies treebank and showed that its coverage can still be improved but already covers a majority of the syntactic constructions that were identified in the corpus.

The CALL application is also released as free and open source. It combines grammar-based text input, grammar-based translation exercises and features of gamification in a web application. The nature of this application allows its use in language classes. A preliminary evaluation with Latin students and teachers did not yet give significant results but resulted at least in interest and positive feedback.

Finally, we implemented and released a framework to learn domain-specific grammars from a GF resource grammar and a set of example sentences. Our evaluation shows that already between 5 and 10 example sentences are sufficient to learn an accurate grammar. With the two additional extensions we also show that we can disambiguate constructions such as PP attachment in specific and restricted contexts.

This thesis shows that all the pieces of the separate jigsaw puzzle are constructed and work independently. For future work, theses pieces have to be fitted together and the whole system has to be evaluated.

Chapter 2

Background

The work in this thesis touches on various areas and can be called interdisciplinary, and justifiably so. For that reason I will try to use this chapter to put the work into context. The relevant background includes Latin linguistics and NLP for Latin, computer-assisted language learning, grammar learning and inference as well as the basics of constraint-satisfaction techniques.

2.1 Latin Language and Latin NLP

The first part of this thesis describes a computational grammar for the Latin language. The Latin language is linguistically classified as an Italic language, which is part of the Indo-Germanic language family. This means that similarities for example in the vocabulary of various languages of this family can be found. Examples of that can be seen in Table 2.1.

Latin	Ancient Greek	German	Swedish	English
pater	πατήρ (=patḗr)	Vater	fader (shortened to far)	father
ager	ἀγρός (=agrós)	Acker	åker	acre (obsolete for field)
trēs	τρεῖς (=treîs)	drei	tre	three
decem	δέκᾰ (=déka)	zehn	tio	ten

Table 2.1: Similarities within Indo-Germanic languages

2.1.1 Latin Language

Latin is supposed to have developed from the regional dialect of the city Rome, along other rural dialects of the Latium region. Over the time it overtook the other Italic languages when the Roman empire expanded.

The history of the language can be divided into three major epochs: the early epoch (about 240-80 BC), the classic epoch (about 80 BC to 117 AD, starting with the first public speeches of Cicero) and the post-classic epoch. Usually, the main focus is put onto the classic epoch with a large corpus of well-known literature. The post-classic epoch reaches until the beginning of the 20th century, when Latin was still a relevant language in science and religion.

Relevant for the work on a Latin grammar are two features of the language:

- Latin is a morphologically rich, inflected language
- Latin has an extensive case system and strict agreement between constituents which allow for rather free word order

The first point means that in Latin many linguistic features are directly encoded in a particular word form, where other languages use other means, such as syntactic constructions. For example Latin has an extensive case system and subject as well as direct object are marked with a respective case, ***nominative*** for the subject and ***accusative*** for the direct object. Because of these case markings, the word order is not as important and can be handled in a more flexible way. These two points will be explained in more detail in Section 3.1.

2.1.2 Latin NLP

The Latin language only plays a minor role in the NLP community and NLP only a minor role within Latin linguistics. However, within the related field of generative grammars, there was already some interest in Latin in the 1960s (Mateu and Oniga, 2017). Furthermore, various theories of formal linguistics have been tested on Latin, including combinatory categorial grammars (CCG, Steedman, 2016).

Besides that, computational approaches to certain aspects of Latin have been presented over the time. The main focus has been on morphological analysis. As a result, several morphological analyzers for Latin are available (LEMLAT, Passarotti, Budassi, et al., 2017; LatMor, Springmann, Schmid, and Najock, 2016). Furthermore, digital corpora and lexical resources are available. An earlier approach is the project Perseus (Crane, 2018) which later got included in the well-known Universal Dependencies project (UD, Nivre et al., 2019). A project to collect and combine linguistic resources and NLP tools for Latin is LiLa: Linking Latin (Passarotti, Cecchini, et al., 2019).

The biggest gap which I try to fill with my work, is the lack of a computational grammar for Latin, or any means of syntactic analysis on a wider scale in general. There have been experiments showing that in principle it is possible to use Chomsky's minimalist program to analyze Latin syntax (Sayeed and Szpakowicz, 2004) but it has never been scaled up. In general it is also

possible to train a dependency parser such as the Stanford parser (Chen and Manning, 2014) on the Latin treebanks available in UD, but there seem to be no pre-trained models so far. However, a hybrid approach between rule-based and statistical approach has been suggested by Win Berkelmans.[1]

With our grammar we hope to provide an important resource that can be included both in specific applications and in general research about Latin linguistics by e.g. helping to bootstrap new treebanks.

2.2 Computer-Assisted Language Learning

The history of computer-assisted language learning (CALL) goes back almost as far as the history of modern computers, at least to the middle of the 20th century, some people even choose earlier dates (Computer History Museum, 2010). Its relevance increased in the same way as technology got more accessible, now with ubiquitous computing it has arrived in the mainstream.

2.2.1 History of CALL

It is almost impossible to give a full account of the history of CALL. An overview for the earlier periods of CALL ranging from the 60s to the 90s of the previous century is given by Levy (1997). For each of the three periods 60s/70s, 80s and 90s he presents major projects, their technological background and their influence on the field. Most of these historic developments are described in a similar way by Warschauer (2004), whose classification of the CALL history can be found in Table 2.2. However, Bax (2003) opposes this classification pointing out that most of the systems survived their own epochs in one way or another.

Stage	1970s-1980s: Structural CALL	1980s-1990s: Communicative CALL	21st Century: Integrative CALL
Technology	Mainframe	PCs	Multimedia and Internet
English-Teaching Paradigm	Grammar Translation & Audio-Lingual	Communicate Language Teaching	Content-Based, ESP/EAP
View of Language	Structural (a formal structural system)	Cognitive (a mentally constructed system)	Socio-cognitive (developed in social interaction)
Principal Use of Computers	Drill and Practice	Communicative Exercises	Authentic Discourse
Principal Objective	Accuracy	Fluency	Agency

Table 2.2: Epochs of CALL from the 1960s to the 1990s according to Warschauer (2004)

[1] Unpublished presentation at the International Colloquium on Latin Linguistics (ICLL) 2019, Las Palmas de Gran Canaria

2.2.2 Modern CALL

A look both at the recent publications in CALL-related conferences and workshops such as NLP4CALL,[2] BEA[3] and NLPTEA[4], as well as at the popular apps in the app stores, can give a picture of the current CALL landscape. Besides full-fledged language learning systems like the one presented in this thesis, a lot of current research is in specific sub-problems like learner modeling, automatic essay grading, and many more.

One of the most popular apps, which now is a commercial system that started with a clear crowd-sourcing approach, is *Duolingo* (Garcia, 2013). Furthermore, a variety of specific systems for different languages and language concepts using different technologies developed in recent years:

- Morphology training with finite state methods (Kaya and Eryiğit, 2015)
- Linguistic resources such as annotated data or semantic resources combined with rule-based exercise generation (Moritz et al., 2016; Michaud, 2008; Redkar et al., 2017)
- Linguistic resources combined with both machine learning and rule-base approaches (Volodina et al., 2014)
- Crowd-sourcing in combination with machine learning (Kenji Horie, 2017)

On the other hand, there is a lot of research in supporting technology. These include learner proficiency level prediction (Pilán, Volodina, and Zesch, 2016), assessing complexity levels for vocabulary (Alfter and Volodina, 2018) and collection of learner corpora (Stemle et al., 2019). This kind of research is relevant to CALL in general and can also influence future work in connection with our current approach, but is at the moment of less relevance than the comparison with applications similar to ours.

2.3 Grammatical Framework

A piece of technology that is present in all parts of this thesis is the Grammatical Framework (Ranta, 2009a; Ranta, 2011). Grammatical Framework (GF) is a grammar formalism and grammar development framework suitable for developing computational grammars for natural languages.

The use of GF is by far not a strict requirement and we show in the respective papers under what conditions it can be replaced by equivalent grammar formalism. However, in our work it seemed the most suitable choice.

[2]Natural Language Processing for Computer-Assisted Language Learning `https://spraakbanken.gu.se/eng/research/icall/nlp4call`

[3]Workshop on Innovative Use of NLP for Building Educational Applications `https://sig-edu.org/bea/current`

[4]Workshop on Natural Language Processing Techniques for Educational Applications `http://www.nlptea.org/`

2.3.1 Abstract and Concrete Syntax

From a linguistic point of view, GF is a grammar formalism that describes languages on two levels: on phenogrammatic and on tectogrammatic level, two terms borrowed from logic by Haskell Curry (1961). The phenogrammatic level describes the surface structure of a language while the tectogrammatic level describes an underlying, more abstract level, of the language. Languages can have different surface forms but share the same underlying structure. An example is prefix, postfix and infix notations for arithmetic expressions. They are different on the surface but have the same underlying structure. In a similar way we can express cross-linguistic phenomena for natural languages.

On the other hand, from a computer science point of view GF is a special purpose programming language, syntactically similar to functional programming languages like Haskell, and based on a type system related to constructive type theory (Ranta, 2015). What was called tectogrammatic before is called the abstract syntax and what was phenogrammatic is called a concrete syntax. The abstract syntax provides just the grammatical categories and the type definition for the syntactic functions. For each abstract syntax, which defines an abstract interface, several concrete grammars can be created, that implement this common interface.

The abstract syntax looks similar to context free grammars, with two major differences. There is no clear distinction between terminal and non-terminal symbols and all the rules, also called functions, are labeled. This makes equivalent to many-sorted algebras[5] where the syntactic categories are the sorts and the syntactic functions are functions defined on these sorts.

The concrete syntax on the other hand is a lot like typical functional programs. Each abstract syntactic category is assigned a concrete data type. The most important basic types are strings and string tuples, i.e. compile-time and run-time strings. Besides that, the developer can define finite parameter types by enumerating all values. Finally, complex data types can be created. The two kinds of complex data types are tables and records. Tables are total functions from a parameter type to any other type. And records are labeled unions of types. Typical use of tables is for inflection tables and records can be used to store inherent grammatical features such as noun **gender**. As mentioned before, we do not define terminal symbols separately. Instead we define constant functions, i.e. functions without a parameter such as `green_A` or `Chomsky_PN`.

Here I present GF using the syntax that is used to write grammars and is directly understood by the GF compiler. In several of the articles we used various levels of abstraction and various syntactic approaches to describe GF grammars. For example, treating the abstract syntax as a many-sorted algebra allows us to reason about the grammar similar to reasoning about algebras in mathematics in general. A different syntactic approach is to express GF grammars as attribute-value matrices. This representation is well known within

[5] Many-sorted algebras (Wirsing, 1990, pp. 680) can be defined given a signature $\Sigma = \langle S, F \rangle$ where S is a set of sorts and F is a set of function symbols with a mapping $\mathbf{type} : F : S^* \longmapsto S$ which expresses the *type* of each function symbol.

linguistics and can be easily used for record types and modified to also cover tables. This syntax allows us to present GF to an audience that is more used to other formalisms expressed as attribute-value matrices such as generalized phrase structure grammar (GPSG, Gazdar, 1985), head-driven phrase structure grammar (HPSG, Pollard, 1994) and lexical-functional grammar (LFG, Kaplan and Bresnan, 1982; Bresnan, 2001)

2.3.2 An Example Grammar

Because the previous description of the grammar formalism is very abstract if you are not used to these concepts, I will illustrate it with an example.

```
1  abstract Chomsky = {
2    cat S ; NP ; VP ; PN ; N ; V ; A ; Adv ;
3    fun
4      mkS : NP -> VP -> S ;
5      mkNP : N -> NP ;
6      mkNP' : PN -> NP ;
7      adjNP : A -> NP -> NP ;
8      mkVP : V -> VP ;
9      advVP : VP -> Adv -> VP ;
10     colorless_A : A ;
11     green_A : A ;
12     idea_N : N ;
13     book_N : N ;
14     sleep_V : V ;
15     furiously_Adv : Adv ;
16     Chomsky_PN : PN ;
17 }
```

Listing 2.1: Example of an abstract syntax

Listing 2.1 defines a new abstract syntax with the grammar name `Chomsky`. In line 2 a list of syntactic categories is defined. These categories are then used in the definition of syntactic functions. The first function, for example, defines that noun phrases and verb phrases can be combined to a sentence.

Based on the abstract syntax in Listing 2.2 we can define a concrete implementation for English. For each of the syntactic categories in the abstract syntax we have to define a concrete data type representing constituents of this category. A basic choice is a string representation `{ s : Str }`. To be precise this is a string value contained in a record with the only record field `s`. For other categories we can extend this record with additional fields.

One example where we add additional record fields is the proper noun `PN` which also contains an inherent **number** feature in a record field with the name `n`. As soon as we encounter inflection we can use table types, for example for nouns. Nouns are not constant strings, but their form, in this simple case, depend on **number**. For that reason we use a table type `Number => Str`. Tables can even be combined to form several levels, such as in the definition of verbs and verb phrases. These concrete category definitions are used in the definitions of our syntactic functions.

```
concrete ChomskyEng of Chomsky = {
  param
    Number = Sg | Pl ;
    Person = P1 | P3 ;
  lincat
    S = { s : Str } ;
    N = { s : Number => Str } ;
    PN = { s : Str ; n : Number } ;
    NP = { s : Str ; n : Number ; p : Person } ;
    V, VP = { s : Person => Number => Str } ;
    A, Adv = { s : Str } ;
  lin
    mkS np vp = { s = np.s ++ vp.s ! np.p ! np.n } ;
    mkNP n = { s = n.s ! Pl ; n = Pl ; p = P3 } ;
    mkNP' pn = { s = pn.s ; n = pn.n ; p = P3 } ;
    adjNP adj np = { s = adj.s ++ np.s ; p = np.p ; n = np.n } ;
    mkVP v = v ;
    advVP vp adv = {
      s = table { p => table { n => vp.s ! p ! n ++ adv.s }}
    } ;
    colorless_A = { s = "colorless" } ;
    green_A = { s = "green" } ;
    idea_N = { s = table { Sg => "idea" ; Pl => "ideas" }};
    book_N = { s = table { Sg => "book" ; Pl => "books" }};
    sleep_V = { s = table {
      P1 => table { Sg => "sleep" ; Pl => "sleep" } ;
      P3 => table { Sg => "sleeps"; Pl => "sleep" }}} ;
    furiously_Adv = { s = "furiously" } ;
    Chomsky_PN = { s = "Chomsky" ; n = Sg } ;
}
```

Listing 2.2: Concrete implementation of the `Chomsky` grammar

Some lexical items, such as the adjectives and adverbs, which are not inflected, are defined in a straight-forward way by giving the string literal. Proper names also include an inherent **number** feature; there is just one singular Chomsky. Nouns and verbs can be inflected, nouns by **number** and verbs by **person** as well as **number**. They are defined using the `table` keyword followed by a mapping from all possible parameter values onto the resulting values, in the case of nouns from both **number** values onto string literals. In the case of verbs, two levels of tables are stacked. Verb forms in English depend both on **person** and **number**. To be precise, only the ***third person singular*** has a different form, usually just adding a `"s"`. So we can first analyze the **person** and then we can analyze the **number** parameter and determine the string literal. For the ***first person***, the **number** does not matter and we end up with redundant strings. This can be avoided by using a wildcard matching all remaining cases: the underline character `_`, such as in `table { _ => "sleep" }`.

Finally, we want to combine lexical items, as well as phrases, to form new phrases up to the sentence level. Simple examples are `mkVP` which just copies the value of the verb because verbs and verb phrases are assigned the same data type and `mkNP` as well as `mkNP'` where the information in the lexical items only has to be extended by constant values. Nouns, in our example, are always ***plural*** and noun phrases are always in the ***third person***. The rules `mkVP`, `mkNP` and `mkNP'` form phrases from lexical items.

The remaining rules combine constituents to form larger constituents. To do so we have to be able to access values in records and select values from tables. For example, in `mkS` we need to combine the string of the noun phrase with the correct form of the verb phrase. So we need to select the `s` field in both phrases, which we can do by using the record selection operator `.` (=period), such as in `np.s` and `vp.s`. To combine two strings we can use the concatenation operator `++`[6]. And finally we need to get the correct string literal from the verb phrase. The `s` field in a verb phrase is a table from **person** to **number** to string. Both pieces of information are included in the noun phrase, in the `p` and `n` field. We can use this information together with the table selection operator `!` (=exclamation mark) to retrieve the correct form by writing `vp.s ! np.p ! np.s`. Here we chain two selections because after the first selection `vp.s ! np.p` we end up with a table from **number** to string. The two remaining rules `adjNP` and `advVP` extend a phrase with a lexical item. That means we have to keep the structure of the phrase while modifying the string literals. In the case of `adjNP`, this is rather trivial. We just have to update the string in the `s` field. A shorter solution for this is `np ** { s = adj.s ++ np.s }` using the record extension operator `**` to use the `np` value and only updating the `s` field. A bit more advanced is the case of `advVP` because here we need to change the string literals within a table. That means, for the result we need to build the same tables, but we do not actually care about the parameter values at this point. So, instead of

[6]This operator always introduces a space between the two concatenated strings. This is due to GF's internal handling of compile-time and run-time strings. See Ranta (2011, chapter C.4.8 and C.4.9) for details.

matching the values directly, we can assign to them variable names to reuse later, in this case to select the correct value from the verb phrase. In this situation writing out the complete table keyword can be tedious; we can again take a shortcut writing `{ s = \\p,n => vp.s ! p ! n ++ adv.s }` instead of `{ s = table { p => table { n => vp.s ! p ! n ++ adv.s } } }`.

In a similar way to this English concrete syntax, we can also add a concrete syntax for German or Latin and use GF to translate between these languages.

2.3.3 The Resource Grammar Library

A final, and very important, feature of GF is the resource grammar library (RGL, Ranta, 2009b), which is part of the GF distribution. The resource grammar library consists of an extensive abstract syntax and concrete grammars in more than 40 languages implementing it[7]. One of the languages included is the Latin grammar which will be described in detail in this thesis.

The RGL provides a high-level and language independent access to the linguistic information in the various language grammars. For example, in the resource grammar there are various ways to form sentences. One of them is called `PredVP`, the equivalent to `mkS` in our example, forming a sentence from a noun phrase and a verb phrase. But there is also e.g. `ConjS`, which forms a new sentence by forming the conjunction of two or more sentences. On the higher abstraction layer of the RGL both functions are accesible under the name `mkS` with differing parameters. These functions can be used in domain-specific or application grammars without having to know the underlying implementation.

With our grammar we contribute to the GF ecosystem and add Latin to the languages that can be added to applications using the resource grammar library.

2.4 Grammar Learning

In the final part of this thesis we want to learn grammars that we can use in our approach to CALL. There have been, over the time, quite a few approaches to learning grammars. Which of these methods is the most suitable depends on the use case. However, we tried to approach the topic from a broad perspective and narrow it down by checking our requirements. But first I want to present all the approaches we had under consideration.

2.4.1 Grammar Inference

There has been a lot of work on general grammar inference, both using supervised and unsupervised methods (see, e.g. overviews by Clark and Lappin, 2010 and D'Ulizia, Ferri, and Grifoni, 2011). Most of these approaches focused on context-free grammars, but there has also been work on learning grammars in more expressive formalisms (e.g. Clark and Yoshinaka, 2014).

[7] http://www.grammaticalframework.org/lib/doc/status.html

In traditional grammar inference, one starts from a corpus and learns a completely new grammar. Because the only input to the inference algorithm is an unannotated corpus, it can require a larger amount of data to learn a reasonable grammar. Clark reports results on the ATIS corpus of around 740 sentences for one of his experiments (Clark, 2001).

2.4.2 Data-Oriented Parsing

Data-oriented parsing (DOP) (Bod, 1992) is an alternative approach to grammar inference. The grammar is not explicitly inferred, but instead a treebank is seen as an implicit grammar which is directly used by the parser. This parser tries to combine subtrees to find the most probable parse tree. Results are reported in a similar fashion for the Penn Treebank ATIS corpus with 750 trees (Bod and Scha, 1997).

2.4.3 Probabilistic Context-Free Grammars

Another approach that could come to mind when thinking about learning grammars is using probabilistic context-free grammars (PCFG), an extension of context-free grammars where each grammar rule is assigned a probability. Parsing with a PCFG involves finding the most probable parse tree (Manning and Schütze, 1999, Chapter 11). The probabilities for a PCFG can be learned from annotated or unannotated corpora using the Inside-Outside algorithm, an instance of Expectation Maximazition (EM) algorithms (Lari and Young, 1990; Pereira and Schabes, 1992). However, the PCFG approach does not make any statement where the initial grammar is coming from. As a result it also requires the presence of a grammar in the first place or has to be combined with a general grammar inference method. PCFGs work for both formal and natural languages. Experiments include the palindrome language and a subset of the Penn treebank. Pereira and Schabes (1992) show that training on bracketed strings results in significantly better results than training on raw text. They report results on 100 input sentences for the palindrome language and on the ATIS corpus using 700 bracketed sentences.

2.4.4 Subgrammar Extraction

There has also been, in limited scale, previous work on subgrammar extraction (Henschel, 1997; Kešelj and Cercone, 2007). These articles present approaches to extract an application-specific subgrammar from a large-scale grammar focusing on more or less expressive grammar formalisms: CFG, systemic grammars (equivalent to typed unification based grammars) and HPSG.

2.4.5 Logic Approaches

To our knowledge, there have been surprisingly few attempts to use logic or constraint-based approaches, such as theorem proving or constraint optimization, for learning grammars from examples. One exception is work by

Imada and Nakamura (2009), who experiment with Boolean satisfiability (SAT) constraint solvers to learn context-free grammars. However, they report results only for formal languages over $\{a, b\}^*$.

2.4.6 Discussion

The techniques presented in the previous section are more or less closely related to the work in the third part of this thesis. However, there are also more or less strong differences. First we need to specify our use-case: We want to learn a compact grammar from a very small set of examples sentences. The resulting grammar should be suitable for our own, specific computer-assisted language learning system. Instead of starting completely from scratch, we can assume that we already have a comprehensive, suitable resource grammar.

Most of the methods I just presented, require no or very little linguistic information besides the training examples. As a result they tend to require more and more training data to learn more expressive grammars.

- Traditional grammar inference does not expect any additional input besides the example corpus and produces a completely new grammar. A usual approach to grammar inference is to use context distributions as categories in the grammar (Clark, 2000). Given the results by e.g. Clark (2001), it seems very unlikely that we can get satisfactory results from between 10 and 20 input sentences.

- The DOP model is not using explicit grammars at all. This makes it most likely unsuitable for our application. It is still an interesting approach because it has some similarities with our ideas about subgrammar learning presented in this thesis, especially about using subtrees in the learning process. For DOP, using subtrees considerably improved the accuracy. When using unbounded subtrees the parsing accuracy rose from 27% to 64% and the bracketing accuracy increased from 88.1% to 94.8% on the ATIS corpus (Bod and Scha, 1997).

- Both previous approaches to subgrammar extraction either take different kinds of input or enforce different constraints on the resulting grammar. This again means that, even though the idea seems very close to ours, these methods are not really suitable for our use case.

- The previous logic-based technique only focuses on formal languages. Furthermore, it encodes the learning problem as Boolean expression. As a result it is not directly possible to judge if a solution is optimal.

- The PCFG approach, albeit modified, is actually very promising. One issue is the initial grammar, but GF has built-in support for probabilities, so it would be possible to use the RGL. Another thing missing is the algorithm to estimate the probabilities from the examples. There are potentially other issues, including how to choose the rules for the resulting grammar based on their probabilities.

After surveying many of these techniques, we settled on our own approach that fits our use-case perfectly. To our knowledge, no other method would directly provide what we need: a reliable way to restrict a grammar to a subgrammar that has the desired properties. Our requirements are quite specific, however, our approach could also be useful outside our use case within computer-assisted language learning and of general interest.

2.5 Constraint Satisfaction and Optimization

In mathematics, a constraint satisfaction problem is a problem where a set of objects have to fulfill certain constraints. These constraints can, e.g. depending on the problem, be phrased in as expressions of formal logic or as mathematical equations. Constraint satisfaction problems can be extended to constraint optimization problems by adding a measure for optimality.

2.5.1 Constraint Satisfaction

A popular family of constraint problems are SAT (Boolean SATisfiability) problems. The problem is phrased as Boolean expressions involving logic variables and a SAT solver tries to find an assignment of truth values for all variables that makes all formulas true. These problems can be called a constraint satisfaction problem (CSP) because the objective is to find any solution to the problem if one exists.

A simple problem with its solution can be seen in Figure 2.1. The formula (1) requires S_1 and S_2 to be true. Because S_1 is true, so has to be T_1 according to (2). Line (3) forces either T_2 or T_3 to be true, because S_2 is true. We do not know which of them so we can choose randomly and end up with one of the possible solutions.

The variable names are arbitrary at the moment but chosen similarly to the ones we encounter when describing our approach to grammar learning.

2.5.2 Constraint Optimization

Related to CSPs are constraint constraint optimization problems (COP). In addition to the constraints, they also contain an objective function. An objective function is a criterion that tells the solver how "good" a solution is. Instead of finding just any solution, the solver now tries to find an optimal solution based on the measure given by the objective function. The constraints are usually formulated as linear inequalities. Here the variables can have various shapes, including real-valued, integer-valued or 0/1 integer-valued. Depending on the range that is allowed for the variables, the problem can be more or less difficult to solve. The Simplex algorithm (Cormen, 2009, pp. 864–879), to solve real-valued problems, often has polynomial run-time and to find a solution for integer linear programmings is NP-hard. This means that it is considerably easier to find a solution for a real-valued problem than for an integer-valued problem.

In the case of 0/1 integers, i.e. integers that can only either have the value 0 or 1, the similarity to Boolean variables is quite obvious. This similarity also means that Boolean formulas can be expressed as linear inequalities.

A translation of the problem in Figure 2.1 into a COP can be seen in Figure 2.2. Each of the Boolean formulas has been translated into linear inequalities and we assume that every variable can either have the value 0 or 1. In addition to solving the constraint, we want to minimize the sum of the variables R_1, R_2 and R_3. As a result, we get a different solution than before. Instead of T_2, T_3 is now selected. The reason is, that if we choose T_2 we have to choose R_2 and R_3 in addition to R_1 for T_1, which leads to an objective value of 3, but if we instead select T_3 we only have to choose R_3 for T_3 in addition to R_1 for T_1. This gives an objective value of 2, which is preferable.

Constraint problems are not that common within NLP. However, certain tasks such as word alignment or morphological segmentation (Lilliesträm, Claessen, and Smallbone, 2019) can be modeled as such.

In the third part of this thesis we will be using constraint optimization techniques to create a domain-specific grammar from a resource grammar and a set of examples. The problem can be expressed as SAT, however we also want to add an objective function to get an optimal solution.

Problem:

$$S_1 \wedge S_2 \quad (1)$$
$$S_1 \Rightarrow T_1 \quad (2)$$
$$S_2 \Rightarrow T_2 \vee T_3 \quad (3)$$
$$T_1 \Rightarrow R_1 \quad (4)$$
$$T_2 \Rightarrow R_2 \wedge R_3 \quad (5)$$
$$T_3 \Rightarrow R_1 \wedge R_3 \quad (6)$$

Solution:

$$f(S_1) = f(S_2) = true$$
$$f(T_1) = f(T_2) = true$$
$$f(T_3) = false$$
$$f(R_1) = f(R_2) = f(R_3) = true$$

Figure 2.1: Example of a SAT problem

Minimize: $R_1 + R_2 + R_3$
Subject to:

$$S_1 + S_2 \geq 2 \quad (\equiv S_1 \wedge S_2) \quad (1)$$
$$T_1 - S_1 \geq 0 \quad (\equiv S_1 \Rightarrow T_1) \quad (2)$$
$$T_2 + T_3 - S_2 \geq 0 \quad (\equiv S_2 \Rightarrow T_2 \vee T_3) \quad (3)$$
$$R_1 - T_1 \geq 0 \quad (\equiv T_1 \Rightarrow R_1) \quad (4)$$
$$R_2 + R_3 - 2T_2 \geq 0 \quad (\equiv T_2 \Rightarrow R_2 \wedge R_3) \quad (5)$$
$$R_1 + R_3 - 2T_3 \geq 0 \quad (\equiv T_3 \Rightarrow R_1 \wedge R_3) \quad (6)$$

Solution:

$$f(S_1) = f(S_2) = 1$$
$$f(T_1) = f(T_3) = 1$$
$$f(T_2) = 0$$
$$f(R_1) = f(R_3) = 1$$
$$f(R_2) = 0;$$

Figure 2.2: Constraint optimization problem based on the SAT problem in Figure 2.1

Chapter 3

Overview

In the introduction in Chapter 1 I presented a quick outline of the work contained in this thesis, followed by the background necessary to put the work in this thesis into context in Chapter 2. In this chapter I will dive deeper into the different topics relevant to the three parts of this thesis and give an overview of the work as well as the results and contributions.

Part I focuses on the Latin resource grammar. It contains both a description of how the grammar was implemented and how such a grammar can be tested and evaluated using other language resources. The grammar itself forms the foundation of our Latin language learning application. It provides the necessary language description that can be used in formalizing language learning lessons. Consequently, this part aims at answering the first research question about *Latin NLP and grammar development*.

The second part is dedicated to the description of the CALL application we designed. It demonstrates how CNLs and computational grammars can be utilized in language learning applications. This includes how grammars can be used to steer the user input and how they can be employed to formalize learning objectives for translation exercises. The aim of this part is to answer the second research about *Computer-assisted language learning*.

Finally, Part III ties in between the other two, even though it presents a potentially more generally applicable technique. It describes a method to learn domain- and application-specific grammars from a small set of example sentences. It does so by extracting relevant linguistic information from a resource grammar. Part I describes such a resource grammar and Part II an application that is based on domain-specific grammars. So, it is a fit for our application and as such answers the third research question about *Subgrammar extraction and grammar learning*.

3.1 Part I: Latin Resource Grammar

The first, and oldest, part included in the thesis is the description of a computational Latin grammar. The implementation of this grammar has been ongoing work, already starting in 2013, well before the start of my PhD studies.

However, the effort put into it increased when we started using it as a language resource in our language learning application. The most recent contribution is in the form of an evaluation to assess the quality of the grammar. The evaluation also provides a prototype for general testing of resource grammars, which has been an open issue within the GF community.

3.1.1 Implementation

The implementation of a resource grammar consists of several parts, some of which are more or less important and challenging, depending on the language. The three most relevant parts for Latin are: morphology, lexicon and syntax. The morphology takes a large part of the work, the lexicon only poses some challenges involving modern concepts and the main challenge of the syntax is the handling of free word order phenomena.

3.1.1.1 Morphology

Previously we identified Latin as an inflected language, also called a fusion language. That means that, other than with agglutinative languages, each inflectional morpheme can encode several features at the same time. For example for the verb *audire* we can have the verb form *audi-o* with the stem *audi-* and the suffix *-o*. In this case, the suffix encodes the following features: ***first person***, ***singular***, ***present***, ***indicative***, ***active***.

Latin morphology is considered quite difficult because most of the parts of speech are inflected and can have many different forms. On the other hand these inflections are quite regular and there are fewer exceptions compared to other languages.

To start with nouns, they are inflected by **number** and **case** and have an inherent **gender**. The Latin cases are ***nominative***, ***genitive***, ***dative***, ***accusative***, ***ablative*** and traces of a ***vocative***, which is only expressed by some male nouns of the second noun declension. **Number** can be ***singular*** and ***plural***.

	First Declension (Feminine)		Second Declension (Masculine)	
Case	Singular	Plural	Singular	Plural
Nominative	*grammatica*	*grammaticae*	*liber*	*librī*
Genitive	*grammaticae*	*grammaticārum*	*librī*	*librōrum*
Dative	*grammaticae*	*grammaticīs*	*librō*	*librīs*
Accusative	*grammaticam*	*grammaticās*	*librum*	*librōs*
Ablative	*grammaticā*	*grammaticīs*	*librō*	*librīs*
Vocative	*grammatica*	*grammaticae*	*liber*	*librī*

Table 3.1: Example for the first two declension classes

Nouns in Latin are grouped into declension classes, depending on the inflection pattern used, i.e. which suffixes are used for each **number** and **case** combination. The noun *grammatica* is in first declension class, which contains

almost exclusively female nouns. The noun *liber* belongs to the second class, which contains both ***masculine*** and ***neuter*** nouns. Both classes contain mostly regular nouns, the same as the fourth and fifth declension. The third class is a collection of more irregular nouns.

Adjectives are inflected similar to nouns, except that they do not have an inherent **gender** but instead are also inflected by **gender** to be able to agree with the noun that is modified. Most adjectives also have three different comparison levels (positive, comparative, superlative).[1]

A more complex part of speech is verbs. Verb inflection involves many more parameters. In total there can be more than 96 finite verb forms, depending on **person** (***first***, ***second***, ***third***), **number** (***singular***, ***plural***), **tense** (***present***, ***imperfect***, ***perfect***, ***future***), **voice** (***active***, ***passive***) and **mood** (***indicative***, ***subjunctive***). Besides the finite verb form there are various nominal forms including infinitive, participles, etc. Despite that, most of the verb inflection is very regular. There are three different inflection classes for verbs, two of which contain mostly regular verbs.

Morphology is usually presented in grammar books as inflection tables. We can use the `table` constructs in GF to implement morphology in a similar way and define the noun *grammatica* in the way seen in Listing 3.1[2].

```
1  grammatica_N =
2    table {
3      Sg => table {
4        Nom => "grammatica" ;
5        Gen => "grammaticae" ;
6        Dat => "grammaticae" ;
7        Acc => "grammaticam" ;
8        Abl => "grammatica" ;
9        Voc => "grammatica"
10     } ;
11     Pl => table {
12       Nom => "grammaticae" ;
13       Gen => "grammaticarum" ;
14       Dat => "grammaticis" ;
15       Acc => "grammaticas" ;
16       Abl => "grammaticis" ;
17       Voc => "grammaticae" ;
18     }
19   }
```

Listing 3.1: GF table representation for the lexical item `grammatica_N`

And we could do the same for *liber*, and for any other noun in our lexicon. But a better way would be to have a generic function that automatically generates such a table for a noun. To accomplish this we can first rely on the inflection classes that determine which suffixes have to be attached to the word stem to form a certain inflected form. And on top of the inflection classes we can have some heuristics that chooses the correct inflection class for each

[1] Some adjectives form comparison levels with the help of adverbials

[2] In the Latin grammar we don't mark vowel length

noun in the lexicon. Such heuristics is called a smart paradigm in the GF terminology (Détrez and Ranta, 2012).

For a start we can define a function that implements the first declension, the inflection class the noun *grammatica* belongs to. We want a function that takes the string `"grammatica"` and returns the GF table in Listing 3.1.

Such a function is the function `declension1` in Listing 3.2. This function uses pattern matching on the string to first check if the noun qualifies to be in the first declension and at the same time removes the ***nominative singular*** suffix `"a"`. The resulting stem is stored in the variable `grammatic` which then can be used to form the whole table.

The second step is to add the smart paradigm that automatically recognizes the declension class based on the information in the lexicon. Such a function is `smartNoun` in Listing 3.3. It uses noun suffixes of ***nominative singular*** forms to decide which declension should be used to generate the paradigm. However, except for declension one, two and five, the suffixes are not sufficiently unique to decide if for example *casus* belongs to the second or fourth class. Also, only for the first two classes it is possible to reliably infer the predominant **gender** of the declension class from the noun form. For the fourth and fifth declension class the **gender** can be inferred in many cases but not in all. The text following the double-dash (`--`) are comments that are only intended to clarify the code to the reader and are ignored by GF.

For the third and fourth declension class and all disgressive nouns, we need to include more information in the lexicon to infer the paradigm and **gender** correctly. For nouns we can cover almost all remaining cases by looking at the ***nominative singular*** form, ***genitive singular*** form and the **gender**. So we need to add a second function such as `smartNoun2` (Listing 3.4) that does exactly that job.

In a very similar way we can handle all others lexical items such as verbs and adjectives. In some cases it is even possible to reuse parts of inflections of different parts of speech, such as noun inflection for regular adjectives and both noun and adjective inflection for participles and gerundives.

3.1.1.2 Lexicon

The RGL includes a basic lexicon consisting of about 250 concepts. These concepts are mostly based on a list of most common words in the English language.

It is a challenge in general to translate concepts between different languages. What makes this task even more challenging is when one of the languages is a modern language and the other is a historic languages.

Sometimes it is possible to find equivalent or sufficiently close concepts such as *calceus* for eng. *boot*, which is the closer candidate than *soleae*, eng. *sandals* because they cover the whole foot and not just the soles. But other candidates would be possible as well and potentially more appropriate.

An even bigger issue is when we need to translate modern concepts into a historic language. This basically leaves us with two options: paraphrasing or inventing a “new” word in the historic language, which could either be a

```
declension1 noun =
  case noun of {
    grammatic + "a" =>
      table {
        Sg => table {
          Nom => grammatic + "a" ;
          Gen => grammatic + "ae" ;
          Dat => grammatic + "ae" ;
          Acc => grammatic + "am" ;
          Abl => grammatic + "a" ;
          Voc => grammatic + "a"
        };
        Pl => table {
          Nom => grammatic + "ae" ;
          Gen => grammatic + "arum" ;
          Dat => grammatic + "is" ;
          Acc => grammatic + "as" ;
          Abl => grammatic + "is" ;
          Voc => grammatic + "ae" ;
        }
     } ;
     _ => error "Not first declension"
  } ;
```

Listing 3.2: GF function to apply first declension

```
smartNoun noun =
  case noun of {
    terr + "a" => declension1 noun ; -- Gender feminine
    hort + "us" => declension2 noun ; -- Gender masculine
    verb + "um" => declension2 noun ; -- Gender neuter
    ag + "er" => declension2 noun ;   -- Gender masculine
    -- Already covered by hort + "us":
    -- cas + "us" => declension4 noun ;
    corn + "u" => declension4 noun ;  -- Gender neuter
    r + "es" => declension5 noun      -- Gender feminine
}
```

Listing 3.3: Simplified smart paradigm for nouns based on ***nominative singular*** form. Some nouns of the fourth declension cannot be handled here

```
smartNoun2 nounNom nounGen gender =
  case <nounNom,nounGen> of {
    <pat + "er", patr + "is"> =>
      declension3consonant nounNom nounGen gender
    <nav + "is", nav + "is"> =>
      declension3i nounNom nounGen gender
    <cas + "us", cas + "us"> =>
      declension4 nounNom nounGen gender
}
```

Listing 3.4: Second smart paradigm based on ***nominative*** and ***genitive singular*** forms

back-translation from a related modern language or a completely new invention. When developing a grammar, like I did with the Latin grammar, none of this should be directly our job. Studying lexical items is the focus of lexicographers. Instead we would like to rely on suitable external resources.

These lexical resources can be historic Latin dictionaries, modern dictionaries published for students (e.g. Vilborg, 2009) or modern online resources. The best online resources available are crowd-sourcing-based project such as Wiktionary and Wikipedia. The English Wiktionary offers translations for many concepts that often also include Latin translations. Wikipedia on the other hand has a complete instance available in Latin with more than 130 000 pages (Vicipaedia, 2020). By cross-referencing between the various languages, Wikipedia can be a useful lexical resource as well.

The problem with both resources is that they are user created for human use and it can be challenging to automatically extract the intended information. For example in Wiktionary, markup is almost exclusively used to make the entries human readable, not to make them machine readable. Another problem is the reliability of the information found in user-generated content. Besides the named problems, these two projects provide invaluable resources. And it can be quite surprising how many modern concepts have a Latin Wikipedia page.

One remaining challenge is the case where we cannot give a translation in a single word but have to paraphrase a concept using a larger phrase. The major problem with that kind of paraphrase is that the types for lexical items and phrases are often not directly compatible and ways to make them work together have to be found.

3.1.1.3 Syntax

The third and final step in implementing a resource grammar is the implementation of the syntactic rules that build larger phrases and constituents from more basic ones.

The implementation of the Latin syntax is guided by two aspects, the Latin language and the syntactic functions defined in the RGL abstract syntax. As a consequence, certain concepts have to be implemented in the Latin resource grammar which are not explicitly expressed in Latin. This is for example the case for definitive and indefinite article. Despite Latin not making any distinction and not even expressing the concept, it is necessary to implement empty placeholders for both to be compliant with the RGL.

The big challenge of Latin syntax is its comparatively free word order. In reality the word order is very much restricted by factors or pragmatics. And there usually have been predominant word orders, depending on the epoch and the genre. But looking at it from a purely syntactic point of view we have to be able to cover all word orders that are possible, both in theory and practice.

The reason why more free word order is possible in Latin than in other languages is that Latin encodes relevant information using morphology and enforces agreement between all components of a phrase. While, for example, in some languages the subject of a sentence has to be in a certain position, in

Latin it is marked with nominative case. And usually the nominative noun phrase is the subject of a sentence, no matter where it appears. In addition it is often possible to see to which noun an adjective belongs, because they have to agree in **gender**, **number** and **case**. This feature, however, is limited by potentially ambiguous word forms.

If we look at the sentence *imperator vetus imperium Romanum imperat* (eng. *the old emperor rules the Roman empire*), *imperator* is unambiguously ***nominative singular*** and agrees with *vetus* in all features. The noun *imperium* could either be ***nominative***, ***accusative*** or ***vocative singular***. The adjective *Romanum* shows agreement features for ***accusative singular neuter***. The verb *imperat* is a transitive verb that requires a ***singular*** subject and a direct object in ***accusative***. The only way to resolve all these constraints is to treat *imperator vetus* as the subject and *imperium Romanum* as the direct object. That also means we can reshuffle the words, for example reorder to *imperator imperat Romanum vetus imperium* and there is still only one possible reading. However, this word order is highly unlikely because there is no semantic or pragmatic reason for it and it would make it unnecessarily difficult to understand the sentence. On the other hand *vetus imperat imperator imperium Romanum* would be more acceptable because it puts a focus on *vetus* (eng. *old*).

Free word order phenomena can occur on several levels, but they can be handled in a very similar way. The most common kind of free word order is the reordering of top-level constituents. Latin had, in the classic epoch, a preference of subject-object-verb. However, almost any other order was possible and can be found, depending on the epoch and author (Bamman and Crane, 2006). As a side note, we also have simpler word orders such as verb-object, object-verb, subject-verb and verb-subject. This is possible because Latin is a pro-drop language, i.e. a language where pronouns in subject position can be dropped since the information is already encoded in the verb. Furthermore, implied or repeated objects can be dropped.

Another free word order phenomenon is the overlapping and interleaving of phrases, called Hyperbaton. A short example can be found in Caesar's Gallic War: *dies appetebat septimus* (Caesar, B.G. 6.35.1). Here the verb *appetebat* (eng. *it was approaching*), appears in the middle of the subject noun phrase *dies septimus* (eng. *the seventh day*). In this simple example only a single word is put in the middle of another phrase, making it discontinuous. It is also possible to have two phrases overlap in a way that both phrases become discontinuous.

Currently there is no easy way to handle free word order phenomena in GF. A generic interleave operator (Ljunglöf, 2004, Section 5.4) would be desirable but has not been implemented yet . The most common way to handle free word order is to introduce a new parameter that determines the word order on a higher level and use record fields to store constituents separately until they can be concatenated in the way defined by the parameter.

On the top level subject, verb and direct objects can be reordered (Listing 3.5). We first define the new parameter with all the possible word orders listed. We introduce two linearization categories, `S` for sentence and `Utt` for utterance. While sentence has the three separate fields `s` for subject, `v` for verb and `o` for object, an utterance is just a single string. To form an utterance from a

```
param Order = SVO | SOV | VSO | VOS | OVS | OSV ;

linat
  S = { s : Str ; v : Str ; o : Str } ;
  Utt = { s : Str } ;

lin
  mkUtt s = combineSentence s SOV ;
  mkUttSVO s = combineSentence s SVO ;

oper
  combineSentence : S -> Order -> { s : Str } ;
  combineSentence snt order =
    { s = table {
      SVO => snt.s ++ snt.v ++ snt.o ;
      SOV => snt.s ++ snt.o ++ snt.v ;
      VSO => snt.v ++ snt.s ++ snt.o ;
      VOS => snt.v ++ snt.o ++ snt.s ;
      OVS => snt.o ++ snt.v ++ snt.s ;
      OSV => snt.o ++ snt.s ++ snt.v ;
      } ! order ;
    } ;
```

Listing 3.5: Functions to reorder subject, verb and direct object depending on a word order parameter

sentence we have to combine the sentence and determine the word order. In the function `mkUtt` we use a helper function called `combineSentence` and set the default word order to subject-object-verb. To also allow for other word orders we can add additional rules such as `mkUttSVO`.

The helper function `combineSentence` is defined with a slightly different syntax. It is defined as an `oper`, a helper function, similar to our more informal `smartNoun`. The definition consists of the name followed by its type. It takes two parameters, a sentence and a word order. The function body first contains the definition of a table, where for each possible value of the word order parameter, the constituents of the sentence are concatenated in the proper order. In the end the function argument is used to select the correct string value from this table.

This combination of parameters to control the word order, records to keep parts separately and tables to finally assemble the parts can be used for basically every aspect of free word order. However, as a result, it requires a large amount of parameters and separate record fields within all the phrase categories to make them potentially discontinuous. So, in theory it is possible to handle every kind of free word order phenomenon, but in practice some compromises between relaxing and restricting the word order have to be taken.

Besides the free word order, we have to take care of agreement between various parts of the sentence. Nouns and adjectives agree in **gender**, **number** and **case**; verbs and noun phrases agree in **number** and **person**. In every sentence we need some component that determines these features for the rest of the sentence.

Nouns have inherent **gender** which is passed on to the noun phrase. **Number** of a noun phrase is in the syntactic theory employed by GF, determined by the determiner, which works well for many languages. The case of a noun phrase is determined either by its syntactic function or by the verb involved. Subject noun phrases are in ***nominative*** and the case of object noun phrases is determined by the verb, in most cases it is ***accusative***.

Finally, we can once again come back to the topic of articles in Latin. These most common determiners are not expressed in Latin but are still necessary to determine the **number** of the noun phrase. This adds to the level of ambiguity when analyzing a sentence because e.g. the sentence *vir vetus est* can mean both *the man is old* or *a man is old*.

3.1.2 Evaluation

In the previous sections I presented important aspects of the implementation of the Latin grammar as part of the RGL. The resulting grammar in its current state can be used in various applications, with our main focus on computer-assisted language learning. This shows its basic practicality but does not necessary say anything about its overall quality.

To fix this issue, I created experiments to assess the quality using available language resources. When analyzing the quality of a resource grammar we can focus on the same three components I presented in the implementation section: morphology, lexicon and syntactic rules. Here the order is slightly different. To be able to properly test a grammar we need access to a sufficiently large lexicon. Based on this lexicon and a corpus we can evaluate the quality of our description of morphology. Finally, we can also conduct some evaluation on the syntactic function.

The primary resource used in the evaluation is the Latin part of the PROIEL treebank (Eckhoff et al., 2018), that is included in the Universal Dependencies (UD) project[3]. For the lexical and morphological analysis we can ignore the dependency trees and only focus on the lexical tokens. For the evaluation of the syntactic rules we can use the dependency trees from the treebank. This can be done in a lexicalized or in a delexicalized way. That means we can replace lexical items by their part of speech (POS) tags or lexical categories before testing the syntax to avoid problems involving lexical coverage and out-of-vocabulary errors.

3.1.2.1 Lexicon and Morphology

To begin the evaluation, it is necessary to add additional lexical resources to the resource grammar, to be able to analyze a reasonable fragment of real-world language data. There are a few lexical resources available for Latin, Wiktionary and Wikipedia have been mentioned before. Another resource would be the historic Latin dictionaries included in the Perseus project (Lewis Ph.D. and Short LL.D., 1879). One drawback all these resources have in common is that they have been created for human users and not necessarily for machine

[3] `https://universaldependencies.org/treebanks/la_proiel/index.html`

```
# newdoc id = Commentarii_belli_Gallici,_Caes.,_Gall._1.1
# source = Commentarii belli Gallici, Caes., Gall. 1.1
# text = Gallia est omnis divisa in partes tres [...]
# sent_id = 52548
1 Gallia  Gallia  PROPN  Ne  Case=Nom|Gender=Fem|[...]       4  nsubj:pass
2 est     sum     AUX    V-  [...]                           4  cop
3 omnis   omnis   DET    Px  [...]                           1  det
4 divisa  divido  VERB   V-  [...]                           0  root
5 in      in      ADP    R-  [...]                           6  case
6 partes  pars    NOUN   Nb  Case=Acc|Gender=Fem|[...]       4  obl
7 tres    tres    NUM    Ma  Case=Acc|Gender=Fem,Masc|[...]  6  nummod
[...]
```

Figure 3.1: Simplified fragment from the Universal Dependencies PROIEL Latin treebank in CONLL format

readability. This means it would involve significant work to adopt the resource to our needs.

On the other hand, there is at least one suitable digital resource. It is a lexicon included in a Latin dictionary lookup and translation tool called Whitacker's Words (Whitaker, 2006). The software is in the public domain and the lexicon is available as a plain text file that can be automatically converted into a GF-compatible lexicon. The resulting lexicon contains 37 404 lemmas out of 39 225 contained in the original lexicon.

After the addition of the large lexicon, it is possible to both analyze the lexical coverage and the quality of the morphology. The information contained in the Universal Dependencies treebank (Listing 3.1) includes both the word forms and the lemmas (second and third column). To test the lexical coverage we can see if we can analyze all the lemmas. To get a basic idea about the quality of the morphological description we can see how many of the word forms we can analyze.

For a more in-depth analysis we need some means of comparison. One solution would be to use the morphological tags that are in included in the treebank (fifth column) as a gold standard. Instead, I also compared against the two state-of-the-art morphological analyzers LEMLAT and LatMor. That way it is possible to compare both the coverage and quality of the morphological analysis. In the cases where my analysis failed I also did a more in-detail analysis. Some obvious flaws could be identified, but in general the results were comparable to the other systems.

3.1.2.2 Syntax

The final part of the analysis was on the level of syntactic functions. Here we can treat the dependency trees as our gold standard. Thanks to the work of Kolachina and Ranta (Kolachina and Ranta, 2016; Ranta and Kolachina, 2017) it is possible to convert between dependency trees and GF abstract syntax trees.

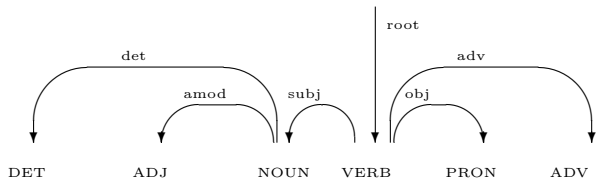
An example can be seen in Figure 3.2. The process involves, as a first step, to replace the POS tags in the UD tree by GF categories. A major problem is that GF categories tend to be more fine-grained than UD POS tags (Figure 3.2b). That means for one UD tree we have to try several candidates for the GF abstract syntax tree. From the categories and the dependency labels involved we can reconstruct abstract syntax functions. For example in the tree we have the categories `AP` and `CN` connected by an edge labeled `amod` which means we can recover the abstract function `AdjCN` (see Figure 3.2c). This conversion can either work on lexicalized or delexicalized trees.

I used this method to convert the trees in the UD treebank into GF abstract syntax trees and analyzed the syntactic functions occurring in these trees. I compared the most commonly occurring functions with the functions missing in the Latin grammar, which currently implements 284 out of 357 functions defined in the resource grammar library. The results show that all of the 20 most frequent functions are already implemented. Besides this result, this experiment also produces a list of most promising rules to be added in a further extension of the grammar.

3.1.3 Results and Contributions

The most obvious result is the existence of a freely available computational grammar. However, building a grammar can be seen as a purely engineering effort. But in addition I conducted a corpus-based evaluation of the grammar. The evaluation shows that there is space for extensions, however, the Latin morphology is able to compete with other morphological analyzers and also the syntactic coverage seems already very well developed. Only a small number of the syntactic rules I could identify in the corpus are missing from the grammar. The evaluation both shows which are the most promising points for future improvement and gives a point of reference for these improvements. All effects of changes to the grammar can be evaluated against the previous version. This allows for an iterative and data-driven improvement of the grammar.

This leads to the contribution in the form of a language resource for Latin, a language outside mainstream NLP. Furthermore, the method used to evaluate the grammar based on other language resources, especially treebanks, can act as a blueprint for testing similar grammars. The evaluation method fills a gap within grammar engineering that has been an issue for years.



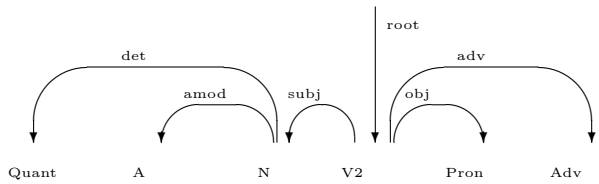
(a) The original dependency tree

UD	GF
DET	Det
ADJ	A,AP,...
NOUN	CN,N,...
VERB	V,V2,...
PRON	Pron
ADV	Adv,AdV,...

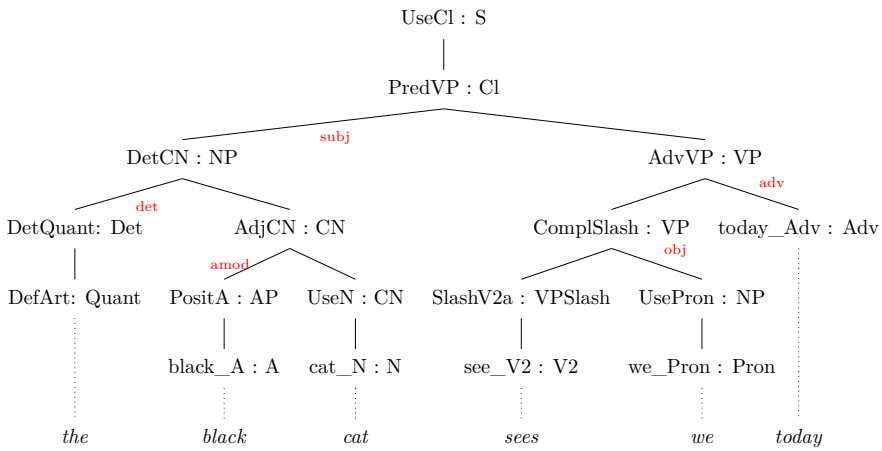
(b) Mapping between UD POS tags and GF categories

Name	Type	Label
AdjCN	`AP -> CN -> CN`	amod head
AdVVP	`AdV -> VP -> VP`	adv head
DetCN	`Det -> CN -> NP`	det head
ComplSlash	`VPSlash -> NP -> VP`	head obj
PredVP	`NP -> VP -> Cl`	subj head
...		

(c) UD labels for GF abstract syntax functions



(d) The dependency tree with correct GF categories



(e) The abstract syntax tree annotated with UD labels

Figure 3.2: Example for the connection between UD trees and GF abstract syntax trees (Kolachina, 2019, p. 11)

3.2 Part II: Grammar-Based CALL

The second part of this thesis, and the part that binds the two other parts together, is a grammar-based application for computer-assisted language learning. The application itself is language-independent and agnostic, however it seemed an obvious choice to test it with Latin based on the Latin grammar described in the previous section.

Our approach to CALL combines an intuitive user input method based on grammars, a theory how to model language learning using specific grammars and other modern features such as gamification. I will present the user input method and the use of grammars for language learning in detail in the following sections.

3.2.1 Grammar-Based User Input

An essential part of our approach to grammar-based language learning is the user interface, especially the input method. It is based on previous work by Ljunglöf (2011). The basic idea is pretty simple. We don't let the user input any text but the system only allows input that is correct according to a grammar. To do that, we do not start from an empty string, but instead start from some grammatically correct sentence. The user can then step by step change the text until it meets their expectation. The usual editing operations are **insertion**, **deletion** and **substitution**, but not on a character level but on a word or phrase level. These operations are implemented by mapping interaction on the surface onto an underlying syntax tree.

insertion One or several words can be inserted by replacing a subtree by a larger tree with more leaf nodes

deletion One or several words can be deleted by replacing a subtree by a smaller tree with fewer leaf nodes

substitution One or several words can be replaced by other words by replacing a subtree by a different tree with the same number of leaf nodes

Subtrees can only be substituted by compatible trees, i.e. the category of the root node of the trees has to be the same. We can demonstrate the method using an example. We start with the almost-Chomskyan sentence *colorless ideas sleep furiously*. It has the abstract syntax tree in Figure 3.3 according to the grammar in Listing 2.1. By clicking onto words we can traverse the tree with a pointer to one of the nodes. The grammar together with the selected node determines the editing operations possible.

In Figure 3.3 we see the movement of the pointer when clicking on the word *ideas* on the surface (blue arrow and circle). This causes the creation of a pointer (red circle) in the tree pointing to the immediate parent introducing the word on the surface, in this case `idea_N`. The system is now looking for all other trees with the same root category `N`. The only other noun in the lexicon is the noun `book_N`. We can now replace *idea* with *book*. If that is not what we

want, we can click on the same word again. This causes the pointer to move one level up to the `mkNP` node. Here the system can offer more alternatives.

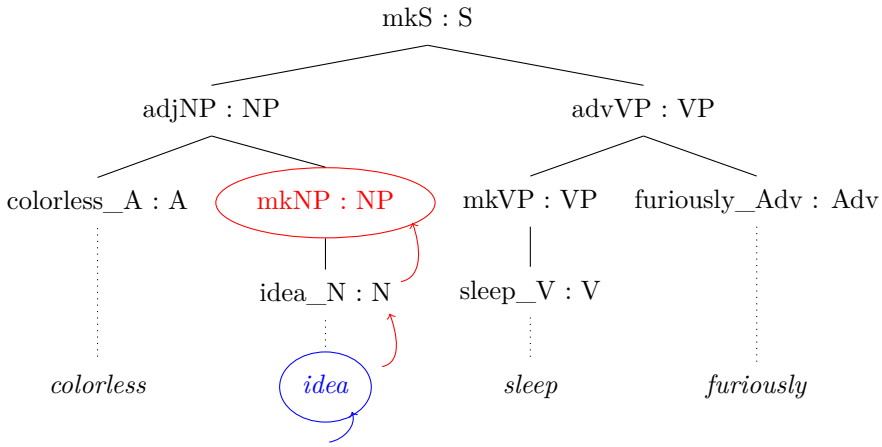


Figure 3.3: Abstract syntax tree for *colorless ideas sleep furiously*. The user clicks on the surface word *ideas* (blue), a pointer in the tree is moved upwards with recurring clicks (red)

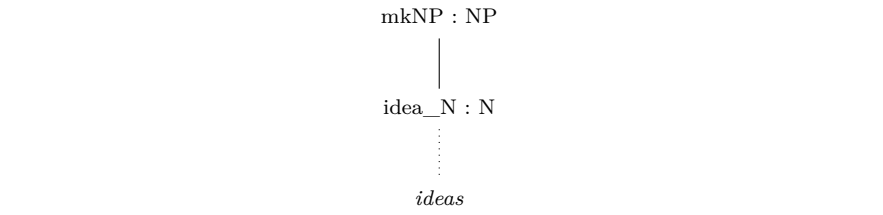
In theory there are infinitely many subtrees of category `NP` but the system filters out certain kinds of subtrees. For example, the original tree will be removed, because it does not cause any change. Also, all trees that can be reached by a sequence of editing steps as well as all trees that can be reached from a node further down are removed.

At the first `NP` node we can get three different subtrees (Figure 3.4b) as replacement for the original tree (Figure 3.4a). They represent the editing operations **insertion** and **substitution**. At this position, no deletion is possible because only one word is highlighted on the surface and removing it would make the sentence ungrammatical.

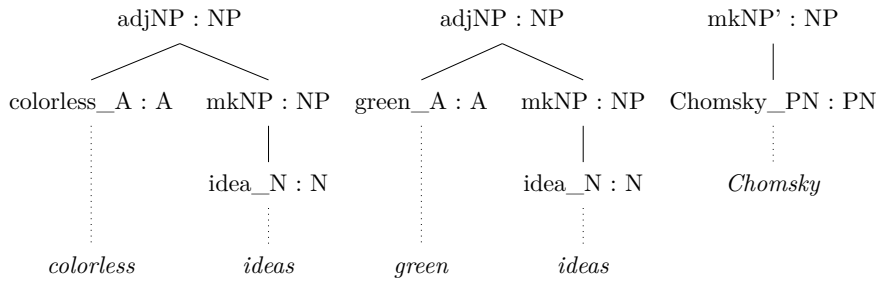
When substituting the subtrees we get the trees shown in Figure 3.5. The result is equivalent to either inserting the adjective *colorless*, inserting the adjective *green* or substituting the proper name *Chomsky* for the noun phrase *ideas*.

The user can now choose one of these edit operations or continue climbing up the tree by clicking onto the same word again. This moves the pointer to the parent node in the tree and starts the process over again. Now we have a larger subtree, covering more words, in focus. This means we can potentially delete words, assuming the grammar allows this. Here it is the case, because we have a smaller subtree with category `NP`. This is equivalent to deleting the adjective *colorless* (Figure 3.6). All other possible subtrees of category `NP` are already covered by the previously covered child and are for that reason ignored here.

In a similar way the user can click on any word on the surface to traverse the tree in the background to find the changes necessary to form the new text



(a) The original subtree



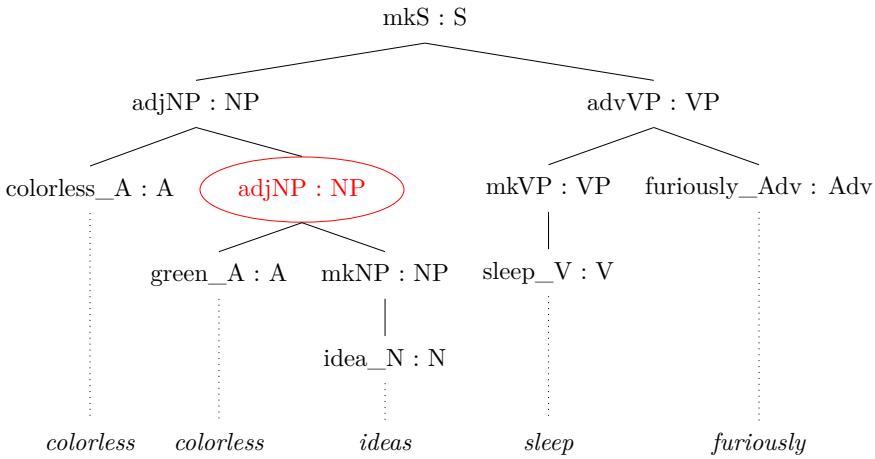
(b) Three potential substitute subtrees

Figure 3.4: The original tree and the accessible subtrees with category `NP`

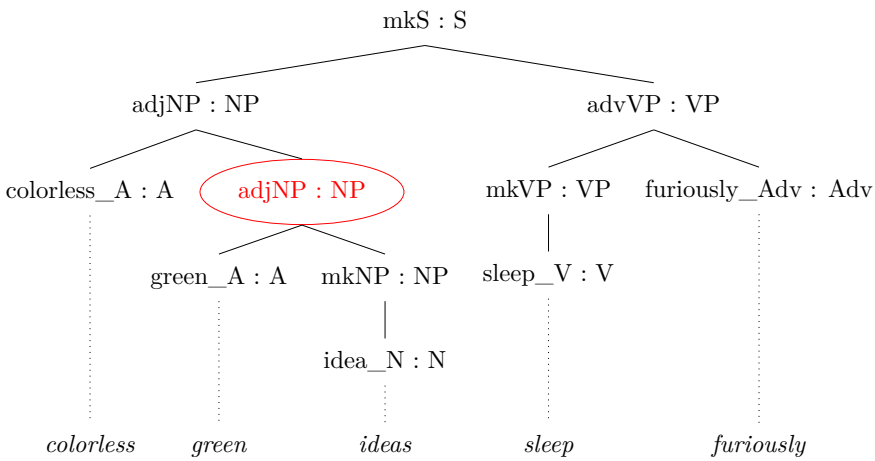
they want to create. The user themself does not have insight into the tree but just selects one of the editing suggestions presented by the system (Figure 3.7). After clicking on a word, the system suggests a list of phrases that can be used in the same place. When selecting one of the suggestions, the phrase is replaced and the process starts over again.

Starting from the sentence, the user can click on any word any number of times. When clicking on the word *ideas* once, the word is highlighted and they get the list of possible replacements, in this case only the word *books*. Clicking on it a second time gives instead: *colorless ideas*, *green ideas* and *Chomsky*. Selecting one of the options changes the sentence accordingly. The grammar guarantees that the resulting sentence remains grammatically correct. For example when choosing *Chomsky*, the system updates the verb form to agree with the ***third person singular*** subject. When clicking on the same word several times, the highlighted area expands and new replacement options are possible. When clicking on a different word instead, the process starts from this selection.

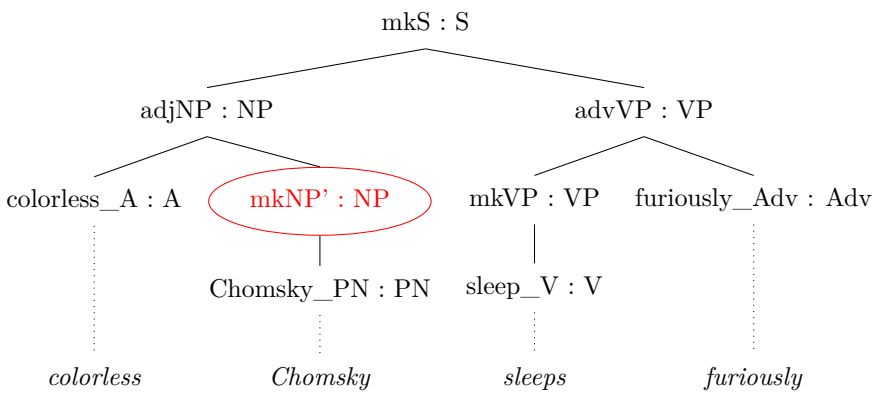
This method provides a generic and intuitive way of editing an input text while at the same time guaranteeing that the result is still within the language defined by a grammar. This makes it a perfect fit for our grammar-based CALL application.



(a) Tree after substituting the first subtree in Figure 3.4, i.e. an insertion

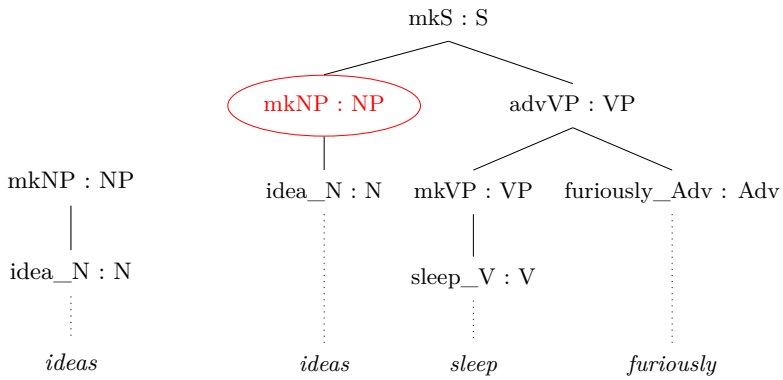


(b) Tree after substituting the second subtree in Figure 3.4, i.e. an insertion



(c) Tree after substituting the third subtree in Figure 3.4, i.e. a substitution

Figure 3.5: The resulting trees after substituting subtrees



(a) The new subtree (b) Tree after substituting the subtree, i.e. a deleteion

Figure 3.6: Editing operation after stepping up in the tree

colorless ideas sleep furiously (Original sentence)

colorless ideas sleep furiously (One click on *ideas*)
Suggestions: Results:
books colorless books sleep furiously

colorless ideas sleep furiously (Two clicks on *ideas*)
Suggestions: Results:
colorless ideas colorless colorless ideas sleep furiously
green ideas colorless green ideas sleep furiously
Chomsky colorless Chomsky sleeps furiously

colorless ideas sleep furiously (Three clicks on *ideas*)
Suggestions: Results:
ideas ideas sleep furiously

Figure 3.7: System interface from the user's perspective when clicking on *ideas*

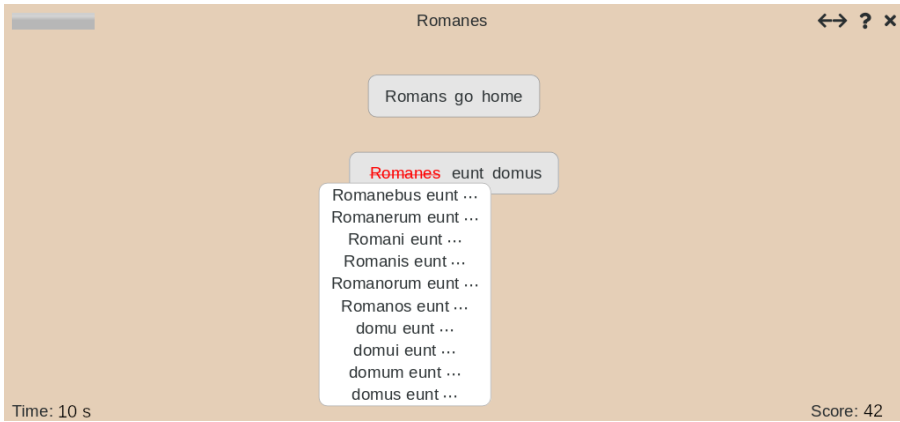


Figure 3.8: The exercise view for a Latin exercise

3.2.2 Grammar-Based Language Learning

Based on the input method described in the previous section, we designed a language learning application. It uses grammars to automatically generate translation exercises on the sentence level.

3.2.2.1 Exercise Type

The next step from the user input to the language learning applications is to define a goal. So far we only looked at editing a single sentence. By using the inherent multilinguality of GF we can add a second sentence in a different language, both sentences being covered by the same abstract syntax. This allows us to easily create translation exercises.

Usually, when learning a second or foreign language, two languages are involved, one already known language used in the learning process and the new language to be learned. Borrowing from logic, we can call the first language the metalanguage and the second language the object language in the learning process.

We start from two different sentences, one in the metalanguage and one in the object language (Figure 3.8). One of the two sentences is fixed, the other one can be edited by the student. Now the task is to change the sentence to make it a proper translation of the other sentence. In the simplest case, each of the two sentences has exactly one abstract syntax tree according to the grammar and the two sentences are translations of each other when the abstract syntax trees match. In fact, because of inherent structural ambiguity, it is more likely that each sentence can be caused by several abstract syntax trees. That means, we have to assign to each sentence the set of all its valid syntax trees. To find out if a sentence is a proper translation of another, we have to check the intersection of the two sets. If the intersection is non-empty, they are translations.

As an additional feature we add color highlighting to show the progress in

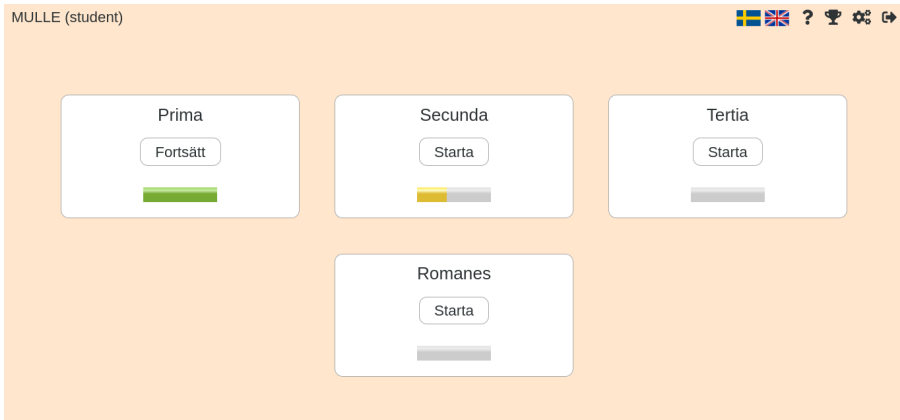


Figure 3.9: The lesson selection screen for a Latin course

the translation task. Already matching parts of the surface string, determined by the same subtree in both trees, can be highlighted to show what parts are already translated and which still need to be translated. In addition, we can highlight different constituents in different colors to highlight the alignment of phrases.

Another important aspect of a language learning exercises is a scoring system. The simplest possible scoring system is simply a timer. The less time it takes to solve an exercise the better. An alternative solution would be to start from a maximum score and subtract points both for time spent and for clicks used. This would encourage both solving the task quickly and accurately.

3.2.2.2 Lesson Structure

Language learning is traditionally subdivided into units, usually lessons and exercises. We can mimic this division by modeling lessons as distinct grammars. An example lesson selection screen can be seen in Figure 3.9. Based on each grammar, a lesson consists of a list of exercises. And each exercise is a pair of two sentences, in different languages, as well as the direction of translation, i.e. which of the two sentences can be changed by the user. A color indicator in the lesson view shows the learner's progress within a lesson, i.e. how many of the exercises are solved successfully. The lesson *Prima* is completely solved (marked in green), *Secunda* is partially solved (marked in yellow) and the other lessons have not been attempted yet.

We want to put a special focus on these grammars that are used to define the lessons. In the GF literature one can often find a distinction between syntactic or resource grammars and semantic or application grammars. The first kind follows the tradition of phrase structure grammars to describe large parts of a language. The categories and functions are linguistically motivated and describe the syntax of the language. Application grammars, on the other hand, use semantically motivated, application specific categories and rules. While the resource grammar might make statements how verbs and nouns can

be combined, an application grammar might speak about combining objects and actions into commands. Application grammars are related to the concept of controlled natural languages (CNLs). CNLs are fragments of natural language with restricted syntax and well-defined semantics. One of the more well known and ambitious CNLs is Attempto Controlled English (ACE) (Fuchs et al., 2005), a large fragment of the English language that can unambiguously be translated into discourse representation structures (Kamp and Reyle, 1993). By definition, each GF grammar can be seen as a controlled natural language because it can only describe a limited fragment of a natural language but at the same time always includes a well-defined semantic representation in the form of the abstract syntax.

We place our lesson grammars between resource grammars and application grammars. They are small compared to both resource grammars and usual application grammars. Their structure is closer to syntactic grammars but we have strong requirements about their semantics. To not confuse the student we want to avoid as much ambiguity as possible, be it structural or lexical. And from our point of view it seems also counter-productive to include completely nonsensical sentences.

It is a challenge to create useful lesson grammars. They can be created manually and particular focus can be put on what grammatical features should be taught in the lesson. This requires an in-depth knowledge of the language, pedagogics and grammar engineering.

A different approach is to adopt the lesson structure from an existing language course. In many language learning contexts, traditional text books are still in use. They are already separated into lessons and each lesson consists of several components: an exercise text, a vocabulary list, some explanation of vocabulary and grammar and some additional exercises. We can base our own lesson grammars on both the text fragments and vocabulary lists provided for each lesson. These grammars can either be created manually or, as shown in the third part of this thesis, learned automatically. By describing exactly the lesson text, we guarantee for a similar level of syntactic complexity. The biggest advantage is that the translation exercises we can create from such a grammar seem familiar to the student, because they share structure and vocabulary with the content of the text book. Located in this familiar setting we can create an arbitrary number of translation exercises by recombining syntactic structures and vocabulary.

This particular focus on mirroring the text book structure also allows for an easier integration of our application into an existing language class in a traditional classroom setting.

3.2.2.3 Gamification

A final step to a proper CALL application is the inclusion of gamification features. After reviewing some literature on the topic we settled on adopting features of GameFlow (Sweetser and Wyeth, 2005) with some modifications inspired by MICE by Lafourcade (described in Fort, Guillaume, and Chastant, 2014, section 4).

The main features of gamification included in our language learning application are the following:

Concentration by minimizing the distraction from the task

Challenge by giving a scoring schema

Control by providing an intuitive way to modify the sentence,

Clear goals by providing a lesson structure

Immediate feedback by using a color schema to highlight progress.

These features allow for a more involving and entertaining language learning experience. This is especially useful for languages which are considered boring by students and where no other computational language learning application exists. Cases have been reported where students attempting to learn Latin or Ancient Greek even suffered from anxiety (A. Takahashi and H. Takahashi, 2015). In these situation a more game- and play-like access to language learning could prevent this issue.

All these techniques, the grammar-based user input, the lesson grammars, and the methods of gamification, were combined into a working prototype for which we received positive feedback from both Latin teachers and students.

3.2.3 Evaluation

I designed an experimental evaluation for our approach to computer-assisted language learning. The experiment design has already been covered in some length in my licentiate thesis (Lange, 2018, chap. 5.3). There I also presented a pilot for this experiment, which did not provide any statistically significant results. For that reason we cannot present any published results about it. However, time moves on, and I can give an updated summary of the experiment design here as well.

The experiment is both within-subject and between-subject and focuses within the language learner on the change in language skills and learner's attitude over the time of the experiment and compares the results from a treatment group with the change in one or more control groups. The questions we want to answer with the experiment are:

- Does using our CALL application have a positive effect on the translation competence of students, compared to traditional teaching techniques and competing computer systems.
- Does using our CALL application have a positive effect on the attitude towards learning a language, here Latin, compared to traditional teaching techniques and competing computer systems.

The original design focused on teaching and learning Latin, but in general the choice of the language does not really influence the design. However, the choice of language has an influence on the potential population of participants.

Relevant aspects for a valid experiment are identified variables, a general setup, the sampling of candidates and the evaluation of relevant factors.

The most informative independent variable is the use of our application compared to just the traditional teaching approach. Other variables are also possible, including the choice of language learning application in general, user input method or user interface.

One problem can be *dislike of computers* which is influenced by the population and can depend on the choice of language. For Latin we noticed an increased reluctance to use computers among the students.

The dependent variables are difficult to measure: The change in learner motivation is very subjective and for measuring a change in learning outcomes a longitudinal study is required. Other language learning systems like Duolingo have been evaluated over a longer period in a closed setting under strongly controlled conditions and supervision (Vesselinov and Grego, 2012). Instead my experiment was planned to be available online and to take four weeks for four lessons.

The participants are asked to answer two questionnaires, one in the beginning to control for the background variables and to give a point of reference for the development of the dependent variables about learner attitude, and another in the end to change in the dependent variables. Additionally, a quick translation test gives indications about the participant's translation competence in the beginning and the end.

For a between-subject evaluation, we have to add control groups. They take part in the same experiment but the usage of our application is replaced by other activities. When I designed the experiment in the first place, it was difficult to find suitable replacement activities. In the meantime Duolingo released its Latin course in August 2019. As a result, even for Latin, we can use Duolingo as a means of comparison.

Depending on the location and the languages involved, the sampling question, i.e. the selection of a suitable group of participants, can be solved differently. With a very limited population, e.g. beginner's students of Latin at a university, we have to use the whole population. Depending on the size of the group, it might even be impossible to split into treatment and control groups. When several similar language classes are at hand, each class can be used as one of the groups. For fairness reasons the groups can also be reshuffled after some time, but that also increases the time necessary for the evaluation.

Preferably, we want to create three groups of participants. One treatment group and two control groups. The treatment group uses our application while the control groups spend the same amount of time with other activities. To see if the usage of computers in general contributes to improvements, one of the control groups only uses the traditional learning approach and uses the time for additional written exercises. A second control group can be used to test if our own approach has advantages over comparable approaches. This group could for example use Duolingo or other competing systems.

For a meaningful experiment we have to guarantee for its suitability: This external and internal validity as well as reliability and replicability (Bryman, 2012, p. 69).

Important for this are:

- Use of control groups
- Control for background variables
- Suitable sampling methods
- Suitable measures for variables and stability of these measures

In experiments involving humans, stability is a challenge. Some measures are by their very nature more stable than others: Translation speed usually only varies within a certain range, but subjective self-assessment can vary a lot between two measurements, depending on external factors. So the challenge is to find a balance between stability and applicability.

Finally, this more quantitative research evaluation can be extended by qualitative methods mostly based on individual interview. The interviews can be supported by stimulated recall (Fox-Turnbull, 2009) together with some of the quantitative information collected.

3.2.4 Results and Contributions

The main result of this work is the creation of a prototype language learning application. It is innovative in the way it uses grammars as the central component and provides interesting features. These include a novel way of user interaction, gamification and adaptability to teaching requirements. Even though I designed an experimental evaluation (Section 3.2.3) I cannot provide any published results here. The pilot I conducted only had less than 10 participants, of which only two finished the experiment.

The contributions go far beyond our attempt to create a completely new language learning application from scratch. Instead the two most relevant contributions are the development of our own type of translation exercises. It is related to Cloze tests (W. L. Taylor, 1953), but also more flexible while on the other hand lacking the practice of language use in a wider context. The second, and probably more important contribution, is our work on the use of computational grammars in language learning in general. This allows to have a single source controlling the user input, the exercise generation and the modeling of the broader learning objective.

3.3 Part III: Learning Domain-Specific Grammars

The final major part of this thesis is a method to automatically learn a domain- or application-specific grammar from few example sentences. This provides a second connection between the Latin grammar and the language learning application. It allows to automatically learn the previously described lesson grammars by selecting the necessary constructions from the Latin resource grammar.

We present two methods, the first one is the basic approach to use constraint solving techniques to select the required constructions to cover examples from a resource grammar. This can be described as subgrammar extraction and there has been some related work discussed previously. For the second method we extend the basic method in two ways: we include negative examples in the learning process and we allow the merging of rules. Especially the second extension moves us away from plain subgrammar extraction towards learning even more specific application grammars.

3.3.1 Subgrammar Extraction

We start from a very general grammar, e.g. a GF resource grammar, and a set of example sentences. We want to learn a new grammar which is an optimal subset of the original grammar that still covers all the examples. The constraint solving techniques used are the ones described in Section 2.5, specifically 0/1 integer linear programming. A schema of the learning process can be seen in Figure 3.10.

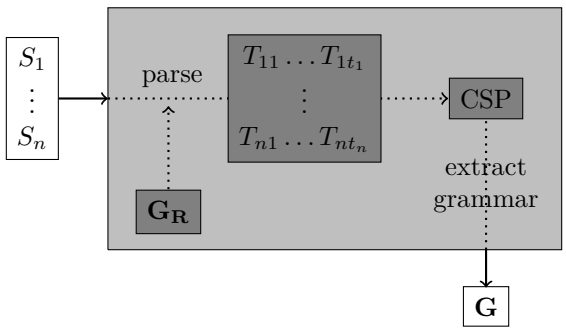
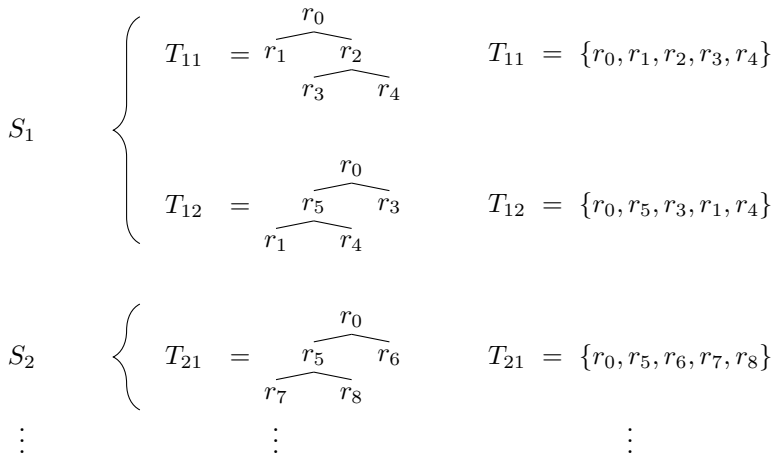


Figure 3.10: The outline of our grammar learning system

3.3.1.1 Modeling as CSP

First we need to describe how we can translate our problem into a constraint satisfaction problem. We use a GF resource grammar as our base grammar. That means we can parse the example sentences with this grammar and get a set of abstract syntax trees for each sentence. Each abstract syntax tree consists of nodes containing the abstract function name and category. We only care about the set of rules used in the tree and ignore both the tree structure and the syntactic categories.

We want to model constraints saying that the set of rules in the resulting grammars still has to cover all the examples. To cover an example at least one of the trees has to be covered and to cover a tree all its syntactic functions have to be covered. We introduce logic variables for sentences, trees and syntactic rules. These variables are used in constraints similar to the ones shown in Figure 3.12.



(a) Sentences (b) Trees (c) Flattened representations

Figure 3.11: Sentences and tree representations

All sentences have to be covered:

$S_1 \wedge S_2 \wedge \ldots$

At least one tree per sentence has to be covered:

$S_1 \rightarrow T_{11} \vee T_{12}$

$S_2 \rightarrow T_{21}$

$\ldots$

All rules in a tree have to be covered:

$T_{11} \rightarrow r_0 \wedge r_1 \wedge r_2 \wedge r_3 \wedge r_4$

$T_{12} \rightarrow r_0 \wedge r_5 \wedge r_3 \wedge r_1 \wedge r_4$

$T_{21} \rightarrow r_0 \wedge r_5 \wedge r_6 \wedge r_7 \wedge r_8$

$\ldots$

Figure 3.12: Encoding Figure 3.11 as logical constraints

As a result we have a conjunction of all sentences, for each sentence a disjunction of trees and for each tree a conjunction of rules. The syntax rules we are using as the atomic units actually are a special case of splitting the tree into subtrees, using subtrees of size 1. Larger subtrees will be a separate topic. These are the logic constraints we use to extract a subgrammar. On top of this constraint satisfaction problem, we can add an optimality criterion to turn it into a constraint optimization problem.

To decide which solution should be considered an optimal solution we experimented with various objective functions. The most promising candidates include:

rules: The sum of rule variables included in the new grammars

trees: The sum of tree variables covered by the new grammar

rules+trees: The sum of both above

weighted rules: The sum of rule variables included in the new grammars weighted by number of occurrences in all trees, preferring more frequently occurring rules

3.3.1.2 Evaluation

To evaluate our method and test the various objective functions we follow D'Ulizia, Ferri, and Grifoni (2011), who present three different evaluation strategies

- "Looks-Good-to-me"
- "Compare-Against-Treebank"
- "Rebuilding-Known-Grammars"

The first one is purely superficial and only checks if the resulting grammars look reasonable. We mostly skipped this test because it is hard to define what *reasonable* means, which makes this test very subjective. Instead we focused on the other two strategies.

The second strategy uses a treebank to evaluate the grammar inference (Figure 3.13). The sentences in the treebank are used to infer a grammar and the trees are used as gold standard to test the quality of the resulting grammar. There are various ways how one can measure the quality of the inferred grammar based on gold standard trees. We chose the simplest case where we just check if the gold standard is among the analyses we get from the new grammar. A more fine-grained analysis would include to also compare subtrees in cases where no complete tree matches. The handcrafted treebanks, contain between 10 and 22 sentences and are available for Finnish, German, Swedish and Spanish.

The final strategy starts from a known subgrammar and tests if we can re-learn the grammar from a set of examples which we can create from it (Figure 3.14). There are again different ways possible to compare two grammars. The

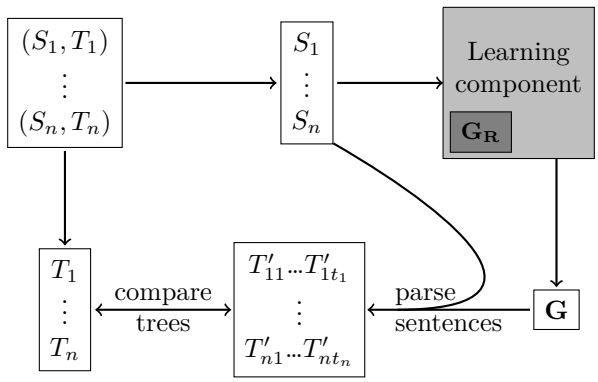


Figure 3.13: Evaluation by comparing the learned grammar to a treebank

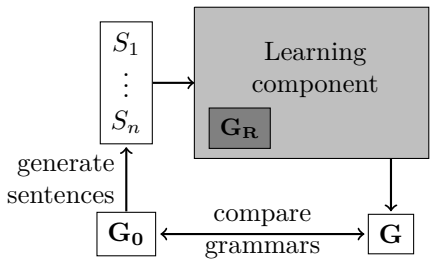


Figure 3.14: Evaluation by rebuilding a known grammar $\mathbf{G_0}$

easiest is to use the measures **precision** and **recall** from information retrieval. We compare the original rules to the inferred rules and determine false positives and false negatives. This gives us an appropriate measure for the quality of the learned grammar. We tried this evaluation with a non-trivial subgrammar of the RGL and English, Finnish, German, Spanish and Swedish.

In general we can learn good grammars from around 10 sentences in average, independent of language and objective functions. However, in the evaluation we encountered some problems with languages such as Finnish, which tend to be more ambiguous and cause more analyses per sentence. In these cases it proved useful to extend from monolingual grammar learning to multilingual grammar learning.

In multilingual subgrammar learning we use translation tuples of sentences as well as the corresponding resource grammars as the input. The learning component is mostly the same. But we take the for each sentence sets of abstract syntax trees for each of the languages and only consider the trees in the intersection. This helps to disambiguate cases of language-specific ambiguity. In our treebank-based evaluation we used translation pairs with English translations and registered improvements for all languages involved.

Despite this evaluation using methods taken from related literature and giving very promising results, we also have to state the limits of our evaluation methods. We evaluated the learning component only with a very limited set of grammars.

For the "Rebuild-Known-Grammar" we used one grammar that is an authentic subset of the GF RGL. The grammar was previously used in some example exercises for the CALL application. We tried to keep this grammar as general as possible while restricting its size. We did not select the rules to improve the results in any way. For the "Compare-Against-Treebank" we basically used whole RGL extended with the necessary lexicon to cover the treebank examples.

As a result, it is difficult to claim that our grammar learning approach is independent from the shape of the grammar and works for all grammars equaly well. However, in our use-case we focus on learning subgrammars of the RGL. And for this we have convincing results.

3.3.2 Beyond Simple Subgrammars

In the previous section I showed how one can learn a application-specific subgrammar from a set of example sentences. This method can be extended in two ways. Before, we were using only positive examples to extract a subgrammar. Our experiments showed that the method provides reasonable results. However, there are good reasons to not just stop there.

3.3.2.1 Negative Examples

The next step is to include negative examples. This allows for an iterative, human-centric learning method. Previously the learning component was basically a black box and it was difficult to directly influence the output. It is

only possible to change the output by providing more examples and hope that this will push the resulting grammar in the right direction. This is fine for a fully-automatic learning method. But to put a human in the loop we need to give them more control over the result.

From a technical point of view it is easy to formulate additional constraints to handle negative examples. Previously we had constraints for positive examples enforcing the presence of all rules to cover at least one tree:

$$
\begin{aligned}
&S_1 \Rightarrow T_{11} \vee \cdots \vee T_{1n} \\
&\cdots \\
&T_{11} \Rightarrow r_1 \wedge \cdots \wedge r_m \\
&\cdots
\end{aligned}
$$

For negative examples we want the opposite. For each negative example, none of the trees should be covered, i.e. for each tree at least one rule has to be excluded:

$$\neg r_1 \vee \neg r_2 \vee \cdots \vee \neg r_m \;\equiv\; \neg(r_1 \wedge r_2 \wedge \cdots \wedge r_m)$$

As a linear constraint it can be formulated as $r_1 + \cdots + r_m < m$. Adding these kinds of constraints to the problem is no problem at all.

Our experiments, both with formal languages and natural language fragments, have shown that negative examples allow us to learn more precise grammars more quickly. This addition also allows us to solve a more general problem, the challenge of including humans in the learning process by creating a process of iterative refinement. In this new process we start with the same automatic process to infer a subgrammar from a list of positive examples we had previously. But now we can use this new grammar to generate new examples and let the person in the loop judge if an example is acceptable or not. This provides us with additional examples, both positive and negative, from which we can learn a new and more accurate grammar.

3.3.2.2 Merging Rules

The second major extension brings us beyond strict subgrammars. From a theoretical point of view, syntactic functions in GF are closely related to the concept of functions in mathematics and computer science. In mathematics if we have compatible functions $f : A \to B$ and $g : B \to C$ we can compose them to a new function $g \circ f : A \to C$. In a similar way we can compose syntactic functions into new syntactic functions.

The next question is, which functions do we want to merge? To find good candidates we can change the basic units in our learning technique from syntax rules to subtrees. This is possible because syntax rules are a special case of subtrees of size 1. Instead of modeling syntactic rules as logic variables we can use the variables to represent subtrees.

Previously, we used three levels to model our constraint satisfaction problem: sentences, trees and rules. Now we add new levels of splits and subtrees. We

split the syntax trees into subtrees up to a certain size in such a way that the subtrees in a split can be reassembled into the original tree. As a result we have the levels: sentence, tree, split and subtrees.

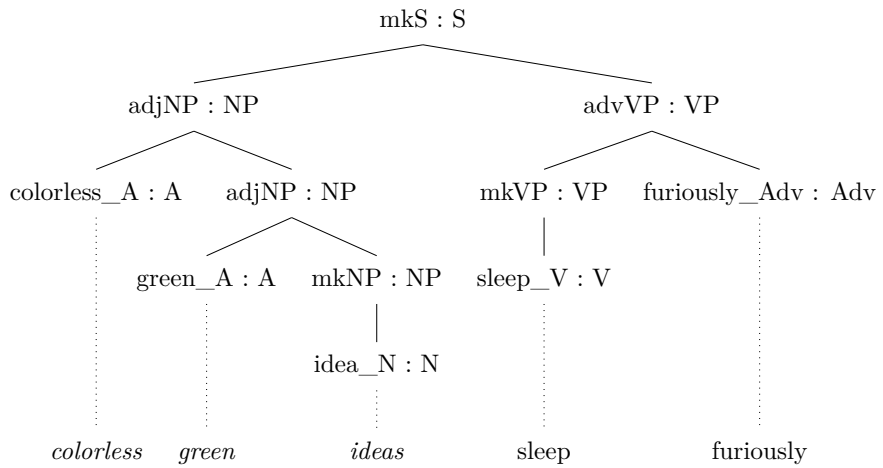


Figure 3.15: Abstract syntax tree for the sentence *colorless green ideas sleep furiously*

If we for example split the tree in Figure 3.15 into subtrees up to a maximum size of 3, we end up with 378 splits, some of which are shown in Figure 3.16. As you can see, the first split is into subtrees of size 1 which is equivalent to splitting into syntactic rules as we did before. This shows that our previous approach is equivalent to subtrees of size 1.

Now we have to integrate the splits into our CSP. Previously, a tree was covered when all it's syntactic functions are covered. Now a tree is covered when at least one of its splits, also called partitions, is covered. The extended CSP can be seen in Figure 3.17.

After solving the COP, based on these new constraints, we can use the subtrees from the solution to merge rules. We testes this method both on formal and natural language fragments and achieved interesting results. For example we could learn the language of balanced parentheses from an over-generating grammar and resolve some attachment ambiguities concerning adverbials.

3.3.2.3 Restricting the Number of Subtrees

In theory, this extension is no problem at all, unfortunately in practice the problem size easily grows beyond what could be considered feasible (Figure 3.18a). For example at subtree size 3 we get 378 splits and 2774 subtrees in total for a tree of size 11. Each subtree has to be represented by a logic variable which means, we quickly and up with millions of variables for longer sentences and several parse trees per sentence.

A working solution is to limit the number of subtrees per split. If we would only allow at most one subtree with size larger than 1 per split in the example

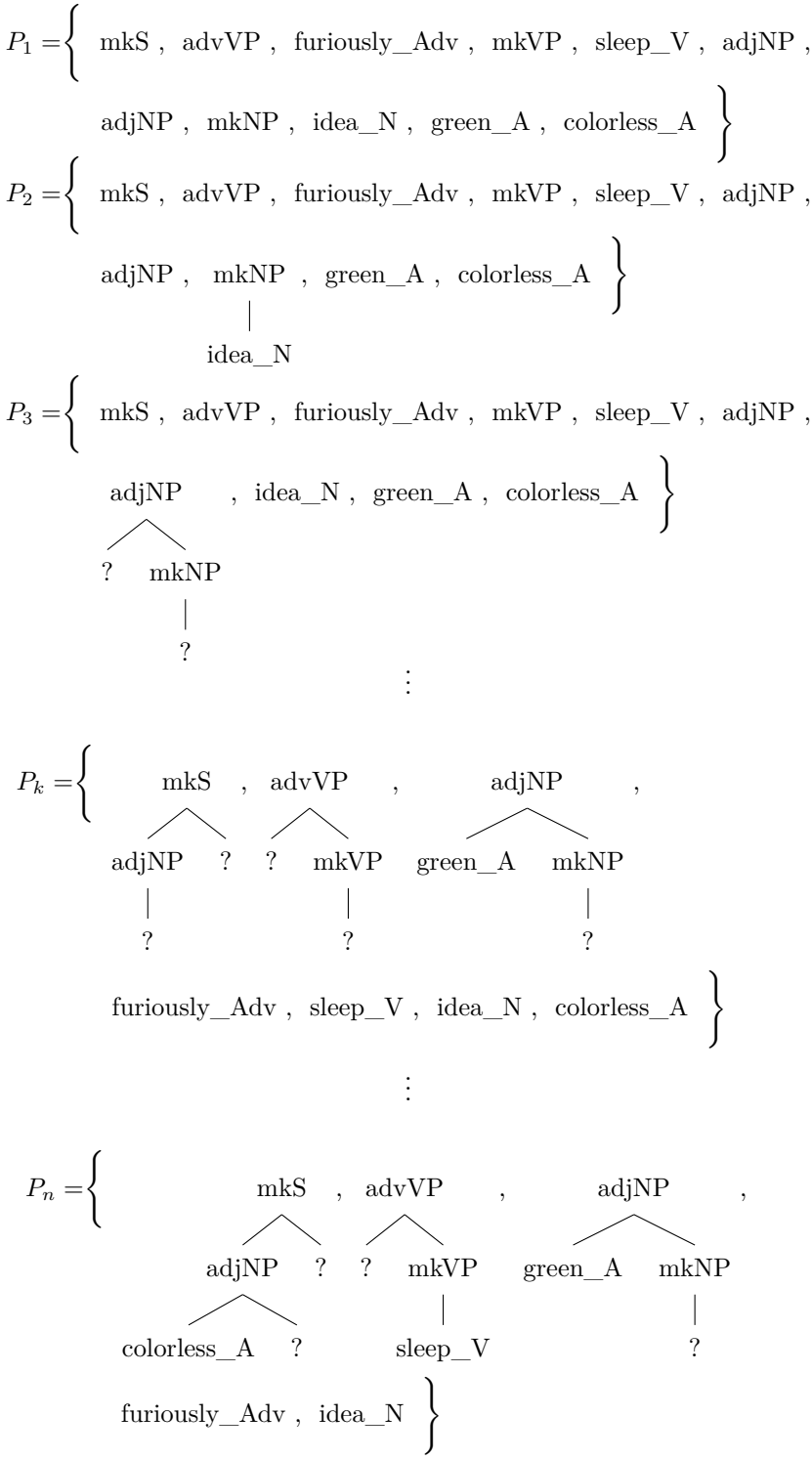


Figure 3.16: Splits into subtrees of maximum size 3 of the tree in Figure 3.15

All sentences have to be covered:

$S_1 \wedge S_2 \wedge \ldots$

At least one tree per sentence has to be covered:

$S_1 \rightarrow T_{11} \vee T_{12}$

$S_2 \rightarrow T_{21}$

$\ldots$

At least one partition per tree has to be covered:

$T_{11} \rightarrow P_{111} \vee \cdots \vee P_{11k}$

$T_{12} \rightarrow P_{121} \vee \cdots \vee P_{12l}$

$T_{21} \rightarrow P_{211} \vee \cdots \vee P_{21m}$

$\ldots$

All subtrees in a partition have to be covered:

$P_{111} \rightarrow st_1 \wedge \cdots \wedge st_n$

$P_{112} \rightarrow st_o \wedge \cdots \wedge st_p$

$\ldots$

Figure 3.17: The constraints including partitions and subtrees

in Figure 3.16, the first three partitions P_1, P_2 and P_3 would be okay but for example partitions P_k and P_n would be discarded because they contain two respectively three larger subtrees.

If we apply this method to the problem of a maximum subtree size of 3 for the same tree in Figure 3.15, we get the resulting graph in Figure 3.18b. It shows that if we allow one or two larger subtrees per split we get substantially lower numbers of partitions and subtrees, and with four or five we get similar numbers as if we would not restrict the number of subtrees.

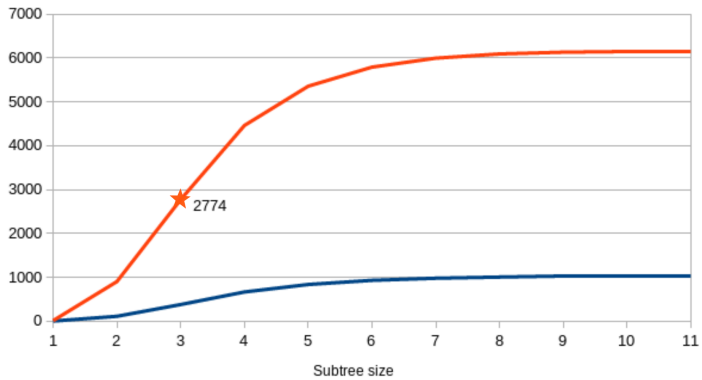
When restricting the number of larger subtrees to 1, we get much more feasible problem and if we run the algorithm repeatedly with this limit, we can merge one subtree into a new rule in each iteration. This can be integrated into the iterative learning method I described previously.

3.3.2.4 Evaluation and Discussion

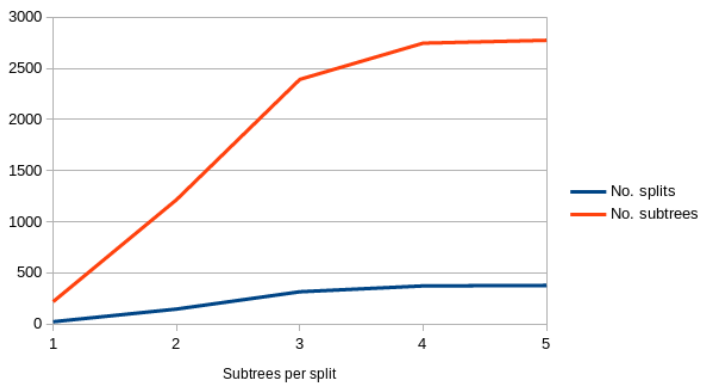
For the extensions of our grammar learning method we don't have an evaluation in the same style as we did for the basic technique. Instead we show on four examples that we can learn interesting grammars. Even though the results are promising, that is more like anecdotal evidence than a proper evaluation.

We can assume that some of the assumption about the basic technique hold, because it is a special case of the extended techniques. One very interesting aspect would be to see how the additions change the learning effect.

The extension of the previous experiments to subtrees is, in theory, trivial.



(a) Number of subtrees and partitions of the tree in Figure 3.15 (tree size 11)



(b) Number of subtrees and partitions when limiting the number of subtrees per partition (tree size 11, subtree size 3)

In practice, however, we have to deal with the aforementioned performance issues. And the mitigation we suggest can again influence the automatic grammar learning method.

To also utilize negative examples would require larger changes to the evaluation methods. One solution would be to modify the "Rebuild-Known-Grammar" in such a way that instead of using two grammars G_0 and $G \subset G_0$ also using $\overline{G} = G_0 \backslash G$ as a third grammar. This third grammar can be used to generate negative examples to be used in the automatic learning experiment. At the moment it is hard to predict how exactly adding negative examples and merging rules will influence the learning of grammars.

3.3.3 Results and Contributions

We implemented the subgrammar extraction method and the evaluation shows that already with about ten input sentences we can learn a reasonable subgrammar. In addition, we can show on examples, using both formal and natural language fragments, that we can learn even more interesting phenomena.

The major contribution as a novel approach to learning grammars by first extracting a domain-specific subgrammar and then optionally merge rules to get an even more specific grammar for a domain given by a small set of example sentences.

Chapter 4

Conclusion

To conclude this first part of the thesis, it is time to summarize and discuss the results. Furthermore, we can give a glimpse at future work and perspectives.

4.1 Summary

The work in this thesis can be seen from two different angles. A narrow angle would be to focus only on our work on Latin and Latin CALL. From this angle, it is easier to answer the research questions.

Grammar development and testing I implemented a computational Latin grammar suitable as a resource for language learning. The thesis contains a description of the necessary steps and methods and demonstrated how its quality can be evaluated using corpus-based methods.

Computer-assisted language learning We created a grammar-based CALL application that uses multilingual controlled natural languages to generate translation exercises. It is based on an intuitive input method that guides the user in the translation task and incorporates features of gamification. The thesis discusses the relevance of grammars and controlled natural languages in the CALL context.

Subgrammar extraction and grammar learning We created a semi-automatic method to generate domain- and application-specific grammars from example sentences. This allows non-linguists to create grammar-based language learning exercises. The thesis demonstrates the feasibility of the learning method.

Some of these results could be criticized for being too much engineering and lacking scientific depth. But there is also a broader perspective to it. Each of the three questions leads to more general contributions in various areas. And each of these contributions help research to progress. Especially the second perspective is relevant to a broader discussion of the results.

4.2 Discussion

Besides just developing a Latin grammar I also explored an approach to handle free-word order languages in general. Together with the experience from handling complex morphology we get closer to best practices for handling languages that are outside the NLP mainstream. And even more importantly, I worked on methods to use treebanks and other linguistic resources to test and evaluate similar computational grammars. This helps move grammar engineering away from the temptation of armchair linguistics towards a stable and empirical foundation of such important resources.

Our contribution to computer-assisted language learning is not just that we worked on Latin, a language that has been outside the mainstream and was in danger of being neglected, even though there is still a significant number of students studying it more or less voluntarily. It is a major undertaking to create a new CALL application from scratch and we did not have the time and manpower to release a nice and polished version. Instead it is still more in the state of a research prototype. But we hope we were still able to enrich the CALL landscape in general.

Our approach nicely fits between purely handcrafted exercises with limited scope and strongly data-driven approaches that provide wide scope but can lead to annoying problems concerning reliability. Also, our exercise type is more interesting than plain vocabulary repetition and more flexible than the related kind of Cloze tests. On the other hand, our approach only focuses on the sentence level and lacks the feature to practice language in a larger context.

Finally, one more relevant point of our approach is its adaptability and flexibility. Our own interest led us to use multilingual grammars implemented in Grammatical Framework and to focus on creating language learning exercises for Latin. But the language learning system is build in a way that the kind of grammars used is not relevant as long as the tree manipulation is possible in a similar way and it is possible to have some form of multilinguality to implement the translations. And even though some challenges are to be expected when adapting the system to other languages, that are conceptually too different, in general it is possible to use any language pairs as long as computational grammars exists for these languages.

For the third part, we of course had our own use case in mind, when we developed it. And we implemented the methods using our favorite grammar formalism. But we also kept the approach as general as possible. This again includes the attempt to minimize the requirements for the grammars involved. As a result, everything that can be formulated as a many-sorted algebra can directly be used as an input grammar.

Still, the importance of this research is hard to estimate. It seems to be more of a corner case compared to other methods of grammar learning such as general grammar inference. Still this does not mean that it is of no scientific interest. And in practice, grammar based applications are not as uncommon as it might seem in a NLP landscape that is mostly dominated by IT giants such as Google, Apple, Facebook and Microsoft. And for everyone involved in developing this kind of applications our research should be very valuable.

4.3 Future Work

It is no surprise that there are still dangling tasks, but many things have already been accomplished. The Latin grammar can still be extended and the corpus-based evaluation has shown where the most relevant errors and omissions lie.

The CALL application can be extended by adding more features, e.g. features of gamification such as competitive features and communication within a class. Or by including related research for learner modeling, example generation, etc. The major remaining endeavor is a proper evaluation of our approach to computer-assisted language learning on a broader scale.

With our method to first learn domain-specific subgrammars and then even merge rules to automatically infer application grammars, a more thorough evaluation seems appropriate. And especially the iterative learning process using negative examples, which we sketched, has to be properly implemented and evaluated.

In general, the majority of the remaining work is to actually assemble the jigsaw puzzle. That means properly hook together the separate pieces of software to build be system we envision. Building and evaluating such a complete system is a separate project on its own.

All this work is again within a narrow perspective. In a broader perspective, I am sure that both our approaches to CALL using CNLs and to learning (sub)grammars are only the start and can lead to interesting lines of future research.

Part II

The Latin Resource Grammar

Chapter 5

Paper I: Implementation of a Latin Grammar in Grammatical Framework

Herbert Lange

Published in , *Proceedings of the 2nd International Conference on Digital Access to Textual Cultural Heritage (DATeCH2017),*
Göttingen, Germany, 2017

Abstract

In this paper we present work in developing a computerized grammar for the Latin language. It demonstrates the principles and challenges in developing a grammar for a natural language in a modern grammar formalism. The grammar presented here provides a useful resource for natural language processing applications in different fields. It can be easily adopted for language learning and use in language technology for cultural heritage like translation applications or to support post-correction of document digitization.

5.1 Introduction

In recent years, the standard approach to natural language processing has been based on statistical methods. Some of the reasons for this focus are that the computing power has increased and, with access to the internet, large amounts of linguistic data have become available. Also statistical approaches already give good results with comparably less effort compared to classic formal and rule-based approaches. But under some circumstances, a manual, rule-based approach might still be considered, e.g. in the creation of language resources for under-resourced languages. Given the fact that languages that are relevant for research in cultural heritage, especially in the Classical Antiquity, still belong to the rather under-resourced languages we present our work in formalizing one of these languages, Latin.

The need for computerized processing for these languages is growing. And our approach can help to achieve that both as a complement and an alternative to statistical methods. Especially a formal grammar can provide some additional insight how a certain language works.

Besides that a grammar gives a stable foundation for different useful applications like Language Learning and providing access to resources in the area of cultural heritage.

One valid point is the question of todays relevance of Latin. But besides the research in cultural heritage, Latin is still seen as the "mother of all Romance languages". Furthermore Latin has a strong relevance since it can be used as a "default" language for linguistic comparisons ("tertium comparationis") (Müller-Lancé, 2006).

Several possible applications can be imagined. A grammar can also be used to generate exercises for language learning applications. That way it is possible to focus on specific characteristics to support the learner in learning certain kinds of syntactic and grammatical constructions.

Another interesting application seems to be the translation of Latin inscriptions and epigraphs. A computerized Latin grammar can be extended with the vocabulary and probably a small set of additional constructions to cover the Epigraphic Database Heidelberg[1]. Together with the possibility to build a Smartphone App, an application can be developed that gives tourists and other interested people the ability to understand Latin inscriptions without having to know Latin.

These applications can also be implemented based on statistical methods, but due to different requirements different kind of models would have to be adopted and trained while the grammar can be adopted without bigger changes.

This paper describes the principle ideas applied in developing this grammar. It starts with some information about the grammar formalism (Section 5.1.1) and the Latin language (Section 5.1.2). The main part of the work is the description of the implementation of the grammar based on a common Latin grammar book (Bayer and Lindauer, 1994). The implementation (Section 5.2) consists of a lexicon (Section 5.2.1), morphology (Section 5.2.2) and syntax (Section 5.2.3). The conclusion (Section 5.3) discusses some future work.

[1] `http://edh-www.adw.uni-heidelberg.de/home/`

5.1.1 Grammatical Framework

The Grammatical Framework (GF) is a modern grammar formalism and a specialized software system for developing of grammars as well as parsing and translation. It is developed as free and open source software at the University of Gothenburg by Aarne Ranta et al. The grammar formalism adopts the style of modern functional programming languages in computer science like Haskell to formalize natural languages (Ranta, 2011).
It is not possible to present a full description of the formalism, so interested readers should be referred to a more comprehensive description e.g. in the book by Ranta (Ranta, 2011). For now it should be sufficient that it implements a variant of context-free grammars extended by additional constructs, most importantly so called tables and records[2]. Adding these increases the expressivity to be equivalent to parallel multiple context-free grammar (PMCFG) (Ljunglöf, 2004).
Tables can be used for parametric features like the noun cases while additional record fields can be used for inherent grammatical features like the noun gender. Parametric features in lexical items usually give rise to inflection tables. In

```
1  -- Possible Number values: Sg (= Singular), Pl(ural)
2  -- Possible Case values: Nom(inative), Acc(usative), Dat(ive),
3  --   Abl(ative) and Voc(ative)
4  -- Possible Gender values: Masc(uline), Fem(inine), Netr (= Neuter)
5  -- Type for Nouns: N = { s : Number => Case => Str ; g : Gender } ;
6    lin
7      man_N = {
8        s = table {
9          Sg => table {
10           Nom => "vir"; Acc => "virum"; Gen => "viri";
11           Dat => "viro"; Abl => "viro"; Voc => "vir" };
12         Pl => table {
13           Nom => "viri"; Acc => "viros"; Gen => "virorum";
14           Dat => "viris"; Abl => "viris"; Voc => "viri"; }
15       } ;
16       g = Masc
17     }
```

Listing 5.1: An example for a noun in this grammar

Listing 5.1 you can see a possible full lexical entry of a noun in this grammar. It shows examples of both tables and records. In the top there is an informal definition of all possible values for several finite features such as number, cases, and genders. This definition is followed by the type for the values that represent nouns in this grammar. It is a record with two fields, the first one, with the label `s`, contains the inflection table for the noun forms and the second field,

[2]Tables types have the form $\{Type_1$ `=>` $Type_2\}$ with $Type_1$ a finite type and $Type_2$ an arbitrary type and values of these types have the form `table {` $k_1 \Rightarrow V_1; \ldots; k_n \Rightarrow V_n$ `}` with k_i of type $Type_1$ and V_i of type Typ_2. Record types have the form $\{l_1 : T_1; \ldots; l_n : T_n\}$ with $l_1, \ldots, l_n$ different labels and $T_1, \ldots, T_n$ arbitrary types. Records values of these types have the form $\{l_1 = v_1; \ldots; l_n = v_n\}$ wit v_i a value of type T_i for every i with $1 \leq i \leq n$. cmp. (Ranta, 2011, pp. 279-282)

with the label `g` stores the inherent gender of a noun.
The inflection table in this implementation is a table of tables, defining that a noun form is dependent both on the number and case.
Finally the full formal definition of a certain noun, the Latin noun *vir*, is given. Here `man_N` is basically an arbitrary identifier, but it is common practice in GF to use a specific format for lexical items. These identifiers have the form of a word-identifying part, usually the English translation, followed by an underscore and the grammatical category.
To this identifier the record value with the two labels `g` and `s` is assigned. The value for the label `g` is simply the gender value `Masc` for ***masculine***. The value for the label `s` however is first a table over the possible number values. Each of this values is again assigned a table over the possible case values. The patterns in the tables must be exhaustive, i.e. there must be an entry in the table for every possible value of the finite feature. The combination of a number and a case value together then determine the word form.
The whole construct after the label `man_N` is the value of a specific noun in this implementation of a GF grammar. The values for other word classes are built in a similar way.
Another characteristic of this formalism is the distinction between so-called abstract and concrete syntax. While the abstract syntax only specifies the rules and their parameters on an abstract level, the concrete syntax gives it a concrete form and specifies how strings are formed according to the grammar. Multiple concrete syntaxes can share the same abstract syntax and the abstract syntax tree can be used as an intermediate representation between the different languages of the concrete syntaxes. The most extensive abstract syntax is the one defined by the resource grammar library (RGL) distributed with GF (Ranta, 2009b). It is already implemented for many languages including the Latin grammar presented here.

5.1.2 The Latin Language

The Latin language belongs to the Indo-Germanic language family and its development spans almost from 240 b.c. to the beginning of the 20th century (Glück, 2004) and in certain fields even continues today.[3] The main focus of this work lies on the Classic Latin period, i.e. the period from the first public speeches of M. Tullius Cicero (ca. 80 b.c.) to ca. 117 a.d., which is also the main focus of common grammar books like the (Bayer and Lindauer, 1994), which was used as the main guideline of this work.
Latin is a language with strong inflection. It belongs to the class of synthetic languages that express syntactic classes and relations through suffixes. In Latin an affix can express several features at once.
Another distinctive feature of Latin is the tendency to a rather free word order. Even though there is still a strong tendency to Subject-Object-Verb word order, it can vary a lot between different text classes and time epochs.

[3] One interesting late instance of Latin in the field of mathematics is Latino sine Flexione (Glück, 2004)

5.2 Implementation of the Grammar

5.2.1 Lexicon

The first part of the grammar implementation described here is a lexicon.
The general plan was to follow the structure of the Latin grammar book (Bayer and Lindauer, 1994). So this section is a bit of an exception since the lexicon is not based on that book, but instead takes the RGL into account, which provides the description of a minimal lexicon containing about 350 lexicon entries.
There are usually a few typical challenges when working with lexicography, most of which we still encounter in this small-scale lexicon.
The first problem is homonymy, i.e. words that share the same spelling but have different meanings. The most common example might be the word *bank*, that has ambiguous meanings in several languages. In this implementation we decided for just one of the possible meanings. Usually other meanings are added to the lexicon with a different abstract identifier. So the translation of `bank_N` would be e.g. the Latin word *argentaria*, the bank that handles money, while the word for the second identifier `bank2_N` would be *ripa*, the bank of a river.
Another challenge are words that have no direct equivalent in the target language so they have to be circumscribed. That leads to larger phrases in the lexicon, that still have to behave like simple words of the appropriate category. So e.g. the translation of the abstract identifier `camera_N` is paraphrased with the Latin phrase *camera photographica*.
The last challenge to be mentioned here is that most of the dictionary entries denote modern concepts. Here the problem is to find plausible translations without inventing new ones. Fortunately there are several collaborative projects like Wikipedia and Wiktionary that provide many useful resources, even for Latin. So e.g. the Latin Wikipedia has more than 100.000 pages (Vicipaedia, 2018) which provide useful information in this task.
Besides adding simple translations, some further decisions have to be made about the lexicon. The most important question is what information has to be spelled out explicitly and what information can be inferred. Our implementation uses a lexicon containing mostly base forms and grammatical information and applies morphological rules to generate the whole paradigms. So in the best case, the lexicon entry just consists of the identifier, one base word form and the grammatical category. However, sometimes it is necessary to add more word forms to generate the whole paradigm for a word or to specify further grammatical information such as case restrictions for the object position of transitive verbs.

5.2.2 Morphology

Word class	Inherent	Parametric	No. of Inflection classes
Noun	Gender	Number, Case	5
Adjective		Degree, Gender, Number, Case	3
Verb		Anteriority, Tense, Number,	4 regular,
(finite,active)		Person	4 deponent
Determiner	Number	Gender, Case	

Table 5.1: Inherent and parametric features for some lexical categories

Feature	Values
Gender	Feminine, Masculine, Neuter
Number	Singular, Plural
Case	Nominative, Genitive, Dative, Accusative, Ablative, Vocative
Degree	Positive, Comparative, Superlative
Anteriority	Anterior, Simultaneous
Tense	Present Indicative, Present Subjunctive, Imperfect Indicative, Imperfect Subjunctive, Future Indicative, Future Subjunctive
Person	1, 2, 3

Table 5.2: Domains of the finite features

This section describes techniques to cope with the challenge of strong inflection in the Latin language and how to generate the whole paradigm, i.e. all possible word forms, for each word from as little information as possible.
The main reason for implementing a rule-based morphology rather than just using a full-form lexicon, is that a lexicon with just base forms is smaller and easier to create and to maintain. It is much more reasonable to store as few as possible forms for each entry and give a set of rules to create the missing word forms.
In Latin there exist several inflection classes[4] for each lexical category. To find the right class for a lexicon entry it is possible to do pattern matching on the base form and apply the rules for the inflection class accordingly. In the terminology of GF this is called a smart paradigm (Ranta, 2011). An example for a smart paradigm for Latin nouns can be seen in Listing 5.2.
Given the great regularity of the Latin morphology, it is possible to reduce the word forms needed in the lexicon to one or just a few. From these the whole paradigm of up to 260[5] forms can be computed. Only for a few exceptions significantly more forms have to be listed, either in the lexicon or hard-coded in the morphological rules.

[4] Classes of words of the same category that construct word forms by a similar schema
[5] All possible verb forms include substantivic and adjectivic forms like gerund, gerundive and supine

The main idea for the implementation of the Latin morphology is to use the construct of tables in GF to list the word forms in tables dependent on grammatical features that are specific for the different word categories. An overview of the parametric features in this grammar can be seen in Table 5.1. The domains of these features are presented in Table 5.2. The inflection tables are created by computing stem forms or other intermediate base forms. To these forms the right suffix according to the features is attached. These suffixes are mostly regular with only a few exceptions.

5.2.2.1 Noun inflection

```
noun : Str -> Noun = \lexform ->
  case lexform of {
    -- noun1, noun2us/um/er, noun4 and noun5 are the functions
    -- for the different declension classes. The 2nd class is
    -- split into three  subclasses
    _ + "a"  => noun1 lexform ;
    _ + "us" => noun2us lexform ;
    _ + "um" => noun2um lexform ;
    -- "Predef.tk n word" removes a suffix of length n from word
    _ + ( "er" | "ir" ) =>
      noun2er lexform ( (Predef.tk 2 lexform) + "ri" ) ;
    _ + "u"  => noun4u lexform ;
    _ + "es" => noun5 lexform ;
    -- Predef.error stops with a given error message
    _  => Predef.error ("3rd declinsion cannot be applied " ++
          "to just one noun form " ++ lexform)
  } ;
```

Listing 5.2: A smart paradigm for nouns in the grammar

Nouns in Latin are, as seen in Table 5.1 declined[6] by case and number while they have an inherent gender. The Latin cases are given in Table 5.2. To enforce vocative as a separate case is an arguable decision since it usually shares the ***nominative singular*** form (Bayer and Lindauer, 1994, p. 21). But the decision to keep all cases has mostly historic reasons.
Given the parametric features and their domains we know that nouns in Latin can have 12 different forms. These forms are created according to five declension classes.
In this grammar we try to use as few word forms as possible. So we implement the smart paradigm for nouns for the first two declension classes, which already cover the majority of the Latin nouns, in a way that one form in the lexicon suffices. The ***nominative singular*** forms of these two classes are quite distinct from the ***nominative singular*** forms which makes it easy to extract the noun stem from them. From this stem we generate all other forms.
The same approach mostly works for nouns of the fourth and fifth declension class, even though some nouns in the fourth declension class have the same suffix in ***nominative singular*** as nouns of the second class. This leads to

[6]Latin noun inflection is called declension

problems in the automatic detection of the declension class but this can be solved by adding additional information to the lexicon.
The most complex case of noun inflection are the nouns of the third declension class. It is so irregular that it is not possible to infer all the important information for inflection and gender from just one noun form. In addition we need to explicitly give more forms as well as the gender of the noun in the lexicon. So we specify not only the ***nominative singular*** but also the ***genitive singular*** form as well as the gender.
Only in rare cases some nouns have so irregular forms that it is easier to list all forms instead of formalizing rules to describe the behavior. On of these nouns we have encountered is the Latin noun *bos* – eng. *cow*.

5.2.2.2 Adjective inflection

The inflectional behavior of adjectives is comparable to the one of nouns. One of the differences is that the inflected word forms are dependent on two more parameters, the gender and the comparison degree. Also there are only three declension classes for Latin adjectives.
In general we reuse rules for noun inflection to create the adjective forms. This is possible since adjectives usually mirror the form of the nouns they agree with. The main difference is that adjectives have to be inflected for all three genders.
Only the comparison levels adds some complexity since some of the adjectives use comparison adverbs instead of dedicated forms for the different levels. That can be handled using a separate record field for these adverbs.

5.2.2.3 Verb inflection

The most challenging word class in terms of inflection are verbs. The basic finite verb forms depend on several features (see Table 5.1). Most of the features shown are well known in Latin linguistics except for the tense system with the distinction into tense and anteriority. This tense system is based on the work of Reichenbach (Reichenbach, 1947, pp. 287-298) and can be uniquely mapped to the traditional Latin tenses.
The number of features involved already leads to a large amount of word forms. Additionally, there are forms derived from verbs that are used in a substantivic or adjectivic way like gerund or gerundive.
The usual way to describe the inflection table for verbs is to use three different verb stems, ***present*** stem, ***perfect*** stem and participle stem (Bayer and Lindauer, 1994, p. 68), modify them according to tense and mood and then attach a suffix that is dependent on person, number and voice. To avoid some complexity we use not only these stems, but also several "base forms" that can be computed directly from the verb form in the lexicon.
Our implementation starts with verbs that can be represented with one verb form in the lexicon. For this base form we decided for the ***present active infinitive*** form. Analogue to the noun inflection the smart paradigm mostly

works with just one base form for all conjugation[7] classes except for the irregular third class.
To handle the complexity of this remaining class as well as special cases in any of the other classes, we again require more verb forms in the lexicon. We decided on ***1st person singular indicative active*** forms in ***present*** and ***perfect tense*** as well as the ***perfect passive*** participle, since these are verb forms listed for irregular verbs in the grammar book.
So we compute the three stems plus base forms for basically every combination of tense and mood. Afterwards the correct suffix for person, number and voice has to be attached.
For the verb forms, that are used like adjectives or nouns, we again reuse the rules for noun and adjective declension.
A special case are the so called deponent verbs, verbs that are used in active voice while their forms are similar to verb forms in passive voice. For these verbs the suffixes have to be adjusted accordingly. Also, since they use passive forms for active voice, they are missing the forms for passive voice (Bayer and Lindauer, 1994, pp. 85-88). The missing verb forms then are marked as non-existent in the paradigm. It is a quite common case that some verbs do not appear in all possible forms.

5.2.2.4 Other Word Classes

Besides these three major word classes which have been presented here the implementation also contains morphological rules for additional word classes. These classes consist mostly of different kinds of pronouns, as well as determiners and quantifiers. Their implementation follows the same principles as we described so far, namely inflection tables and reusage of morphological rules from the other classes if possible.

5.2.3 Syntax

After finishing the morphology we are able to generate all word forms for the entries in the lexicon. The next step is to add syntax rules that use these basic parts to assemble larger parts up to the sentence level. Latin is well known for the flexibility in word order, so we have to consider what parts of a sentence already have to be put together in a fixed order and what should be kept apart.
The technique we can use to keep parts apart as long as necessary are records again. In a record we can keep multiple parts of a phrase separate until we can fix their order. Also we will apply tables again for features that are not fully specified yet and have to be passed on in the syntax tree.
The types of different phrase categories can be seen in the Listing 5.3 together with the necessary parametric features. The keyword `param` indicates the definition of finite features by listing all possible values. In Listing 5.1 we have listed them informally. The first two definitions are special cases because they define new types that depend on values of other finite types. Since

[7] Verb inflection in Latin is called conjugation

```
1  param
2    Agr          = Ag Gender Number Case ;
3    VActForm     = VAct VAnter VTense Number Person ;
4    Anteriority = Simul | Anter ;
5    Tense        = Pres | Past | Fut | Cond ;
6    Polarity     = Pos | Neg ;
7    VQForm       = VQTrue | VQFalse ;
8    Order        = SVO | VSO | VOS | OSV | OVS | SOV ;
9  lincat
10   AP =  { s : Agr => Str } ;
11   CN =  { s : Number => Case => Str ;
12     preap : AP ; postap : AP ;
13     g : Gender ;
14   } ;
15   NP =  { s : Case => Str ;
16     g : Gender ; n : Number ; p : Person
17   } ;
18   VP =  { s : VActForm => VQForm => Str ;
19     obj : Str ; adj : Agr => Str
20   } ;
21   Cl =  { s : Tense => Anteriority => Polarity =>
22     VQForm => Order => Str } ;
23   S =   { s : Str } ;
```

Listing 5.3: Types adjective phrases (APs), common nouns (CNs), noun phrase (NPs), verb phrases (VPs), clauses (Cls) and sentences (S)

Gender has three, **Number** two and **Case** five possible values, the parameter **Agr** for the noun agreement has 3 x 2 x 5 = 30 possible values. **Ag** is not a grammatical feature. It is called a type constructor that binds values of the other three features together as a value of type **Agr**. The verb form **VActForm** is constructed in a similar way.

5.2.3.1 Common Nouns and Adjective Phrases

To form further phrases we start with basic nouns. They can be extended to common nouns (CNs) that can be modified by adjectives.
Since adjectives can be added before and after the noun in Latin, we add fields to store adjectives in the different positions. So CNs can store the noun paradigm, the noun gender as well as several adjective phrases (APs). APs behave just like adjectives except that their comparison level already is fixed.

5.2.3.2 Noun phrases

To create noun phrases (NPs) from CNs we have to add a determiner. Latin in general does not apply the concept of definite or indefinite articles, so these determiners have no surface representation but are still necessary because according to the grammar theory that finds its application in this grammar the determiner determines the grammatical number of a NP (Glück, 2004).
We create the NP by combining the different parts. We first start with a record field that keeps the string the phrase represents. It contains a table

with the parametric feature of **Case**. The NP consists of a determiner, several possible APs and a CN. We first decide that the order of these parts will be the determiner followed by the APs we need to place in front of the noun, then the common noun and finally the APs placed after the noun. We do not enforce that all parts have to exist on the surface. So e.g. there can be a NP that does not contain any APs.
After we fixed the order we have to fix the agreement between these parts. The determiner and the APs have to agree with the noun in gender. The agreement of the determiner is only relevant when we use certain pronouns like e.g. lat. *omnes* – eng. *all* – as determiners. Determiners agree with the noun in gender. Furthermore the noun and the APs must agree with the inherent number feature of the determiner. All parts must agree in the case that is not yet determined but a parametric feature of the NP.
In two other record fields we save the inherent number feature of the determiner and the inherent gender of the CN. Additionally we add a field for the person feature that is fixed to third person. This last feature is necessary for the agreement with the verb form.

5.2.3.3 Verbs and VPs

Now we can take a look at verbs. Verbs can usually be transitive or intransitive. As we have seen before they contain information about lots of possible verb forms. Most of this information is not necessary to build simple verb phrases (VPs).
To construct a VP from a verb we only keep finite ***active*** verb forms. VPs contain two more record fields, one for a direct object and one for an AP.
The field for an AP is used when we want to use an adjective predicatively. The direct object is meant for transitive verbs. For intransitive verbs we do not need the object field and can keep it empty. Also the adjective field will stay empty as long as we do create a VP with an auxiliary verb and an adjective in predicative use.
To fill the remaining fields for the verb forms in a VP we add a parametric feature to the finite forms. It determines if the question suffix `"-ne"` has to be added.
For transitive verbs we do not create VPs directly. Instead we first create an element of the so-called VPSlash-category, i.e. a category that is becomes a VP when a NP is added.[8] So if we combine a VPSlash with a NP we get a complete VP by filling the record field for the direct object.

5.2.3.4 Clauses and Sentences

To form clauses (Cls) we combine a subject NP and a VP. A clause is already quite sentence-like, but several parameters are still flexible. The tense and polarity as well as the word order and the feature about the question suffix of the verb are not fixed yet. These features are again introduced as parametric feature.

[8] in GPSG notation VP/NP, (Ranta, 2011) p. 217

The order of the single parts depends on the feature of the word order. For example in the case of Subject-Verb-Object order we start with the subject NP. It is followed by a negation particle, the AP from the VP and the finite verb form. The last part is the direct object. Again several of this parts can be empty. Other word orders are just reorderings of these parts.
The final step is to ensure the correct forms of all parts. That includes the agreement between several of them. The subject NP has to be in ***nominative*** case. The presence of a negation particle depends on the parametric feature of polarity. The AP must agree with the subject NP in case, number, and gender. The verb form must agree with the subject NP in number. Also the person feature of the verb is decided by the NP. Tense, anteriority, as well as the presence of the question suffix are determined by the parametric features. The object NP is already fixed in all possible features at this point.
As a final step we can fix the remaining parametric features to create a complete sentence (S). In this step we also differentiate into questions and regular sentences. With this step we come to the point where we can analyze or generate Latin sentences with our grammar, the goal we have set for this work.

5.2.3.5 Results

The implementation covers about 530 out of 847 constructions defined in the abstract syntax of the RGL. It consists of 475 lexical rules and 55 syntactic rules. The missing constructions consist of 92 lexical and 225 syntactic rules. The main work was done by one person over a period of about six month.
A full evaluation of the grammar was not possible yet. It is planned for a later point by extending the lexicon with additional resources like Latin-English Dictionary Program Words by William Whitaker[9] and evaluating the coverage of classic Latin texts like Caesar's *Commentarii de Bello Gallico*.

5.3 Conclusion

This work demonstrates the foundations of a rule-based, computerized grammar for the Latin language. We started with a basic lexicon, added a comprehensive implementation of Latin morphology and finally presented rules to assemble larger phrases and sentences from simpler parts.
The grammar can be used to analyze several kinds of sentences including prepositions and questions with a high flexibility in the word order. As usual for grammar development there is always work left to do but the current state is already sufficient to be included in the MUSTE project[10] about word-based text editing.
In the context of this project some of the mentioned applications of the Latin grammar will be explored. A special focus will be on language learning but several other application seem as promising.

[9] `http://archives.nd.edu/whitaker/words.htm`
[10] `http://www.cse.chalmers.se/~peb/muste.html`

Chapter 6

Paper II: An Open-Source Computational Latin Grammar: Overview and Evaluation

Herbert Lange

Submitted to, *Proceedings of the 20th International Colloquium on Latin Linguistics (ICLL 2019), Las Palmas de Gran Canaria, Gran Canaria, 2019*

Abstract

Grammars have, at all times, played an important role in understanding, documenting and teaching languages. This still holds in modern, computer-dominated times, even though the shape of grammars has changed. We present an open source computational grammar, providing a formal description of the Latin language. It covers Latin morphology and syntactic constructions and contains a large lexicon. It is part of a large multilingual grammar framework and based on a modular design, which makes it possible to use it in various application. Examples for such applications are computer-based systems handling Latin language data, including language learning systems and systems for digital access to cultural heritage. Another potential application of a computational grammar is formalizing a linguistic theory that can immediatelly be tested, either by analyzing and but also by generating language data. This article contains both an overview of the grammar itself as well as a description of corpus-based methods usesd to evaluate its quality and coverage. One of the intentions behind this articles is to present the work done in computational linguistics on Latin also to an audience in Latin linguistics, in the hope that they might find it interesting and relevant. The corpora used are Julius Caesar's *Commentarii de Bello Gallico* as well as all other texts that are included in the PROIEL Latin treebank. This choice determines the epoch of Latin in focus to be mostly Classical Latin with the addition of some later texts.

Keywords: Computational linguistics, Latin syntax, Latin morphology, Language resource, Corpus-based evaluation

6.1 The Role of Grammars

Teaching and studying Latin has been an important topic for centuries. In this context, the importance of grammars, i.e. more or less formal descriptions of languages, is generally acknowledged. For the languages of the classical antiquity such as Latin we have traditional grammars that go back for centuries. Over time, many language descriptions have been written and published, traditionally in the form of grammar books. These books have had many roles including teaching the language and defining a "good" style for writers, touching even on the topic of rhetoric. Well-known examples of Latin grammars include *Ars minor* and *Ars major* by Donatus as well as *Institutio grammatica* by Priscian (Murphy, 1980). Other relevant authors include Quintilianus and Varro (Gwynn, 1926; Lehmann, 1934; D. J. Taylor, 1970). The tradition of this kind of language description continues and there are plenty of modern descriptions of the Latin language, of which we can only present a selection (e.g., Pinkster, 1984; Pinkster, 2015; Baños Baños, 2011; Baldi and Cuzzolin, 2009; Baldi and Cuzzolin, 2010a; Baldi and Cuzzolin, 2010b; Baldi and Cuzzolin, 2011). Handling the topic on a smaller scale compared to some of the other grammars mentioned before, school grammar books for Latin are usually used in teaching in combination with a Latin textbook, such as Bayer and Lindauer (1994).

In the 20th century, new formal approaches to linguistics were established, leading to Noam Chomsky's work on generative grammar (Chomsky, 1957). In this movement formalisms were created to describe languages in a rigorous, almost mathematical way. Already from the early days on, formal syntacticians have also been interested in Latin, for example Lakoff as early as 1965 (Mateu and Oniga, 2017). The closeness of these formalisms to mathematics, and consequently computer science, allows for such grammars to be translated into a computer-readable form.

However, there has been a lack of computational grammars for Latin, which have sufficient coverage of the language to be of general use. There has, to the best of our knowledge, only been one computational attempt to handle Latin syntax by Sayeed and Szpakowicz (2004). It tests methods to parse Latin, as an example of a free word-order language, based on Chomsky's Minimalist Program. However, it does not provide a wide-coverage resource and has not been properly evaluated on corpus data on a larger scale.

We fill this gap by providing a computational grammar for Latin implemented in Grammatical Framework (GF), a modern grammar formalism (Ranta, 2009a; Ranta, 2011). The grammar is available as free and open source software in the GF resource grammar library[1], a multilingual grammar resource (Ranta, 2009b). The grammar follows a modular design, combining a morphological subsystem, syntactic rules and lexical resources.

The grammar described in this paper provides a starting point both for building concrete applications and to test linguistic hypotheses about Latin. Following the tradition of grammar use in language teaching and learning, one of the first concrete applications of our grammar is a computer program to

[1] `https://github.com/GrammaticalFramework/gf-rgl/tree/master/src/latin`

practice Latin translation knowledge (Lange and Ljunglöf, 2018a; Lange, 2018).

By including a large lexicon, we can use corpus data to evaluate the grammar. A corpus-based evaluation helps to put the grammar onto a stronger foundation and guarantees that it provides a reliable resource that can be used in various applications. We look at two corpora separately: one is the whole Latin part of the PROIEL treebank (Eckhoff et al., 2018) as it is included in the Universal Dependencies (UD) project (Nivre et al., 2019); the other one is only the part of Julius Caesar's *Commentarii de Bello Gallico* contained in the PROIEL part of the UD treebank.

6.2 Overview of the Grammar

We provide a high-level overview of the grammar and introduce the main concepts used to describe Latin syntax and morphology. It would go beyond the scope of this article to describe both the grammar formalism and the complete grammar in detail. A detailed description of the formalism is given in (Ranta, 2011). Previous descriptions of the Latin grammar include (Lange, 2013), (Lange, 2017) and (Lange, 2018, chap. 4).

6.2.1 The Grammar Formalism

Grammatical Framework (GF) is both a grammar formalism and a special-purpose programming language to describe natural languages (Ranta, 2009a). Grammars are separated into two parts, an abstract syntax and a number of concrete syntaxes. This separation allows for multilingual grammars and consequently for out-of-the-box machine translation.

The abstract syntax describes, on a high level, how smaller parts can be combined to form larger phrases in general, e.g., a noun phrase can be formed from a noun and an adjective. The abstract syntax is language independent.

The concrete syntax is language-specific. For each language that is to be included in a grammar, a concrete syntax has to be defined. It contains the concrete instantiation of the abstract syntax, for example if the adjective should appear before or after the noun.

vīlla f (genitive *vīllae*); first declension

case	***singular***	***plural***
nominative	*vīlla*	*vīllae*
genitive	*vīllae*	*vīllārum*
dative	*vīllae*	*vīllīs*
accusative	*vīllam*	*vīllās*
ablative	*vīllā*	*vīllīs*
vocative	*vīlla*	*vīllae*

Figure 6.1: Lexical information and paradigm for the noun *vīlla*

The basic operation in the grammar formalism is simple string concatenation, i.e. combining a sequence of words to form phrases and sentences. To be able to model more advanced linguistic phenomena, the formalism provides, additional constructions, called tables and records. Tables are mappings from grammatical features to values. The resulting values can either be strings, i.e. words or word forms, or nested tables and records again. An example of the use of tables are inflection tables such as Table 6.1, where noun forms depend on **case** and **number**. They can also be very useful to control the word order (Figure 6.5 in Section 6.2.3).

Records are collections of labeled values. They are conceptually similar to feature structures in other formalisms and are used to store and retrieve separate pieces of information using assigned names. This information can take the form of inherent features, e.g., nominal gender, or store parts of discontinuous constituents and allow reordering of words and phrases.

The Latin grammar described in this article is one of many concrete syntaxes that implement the same abstract syntax. Together these concrete syntaxes form the resource grammar library (RGL) (Ranta, 2009b) which contains an extensive abstract syntax and the concrete syntax for more than 30 languages.

6.2.2 Morphology

Latin morphology is considered difficult to handle because Latin encodes many features on the morpho-syntactic level. It has for example an extensive case system for nouns and inflects verbs by **number**, **person**, **tense**, **mood** and **voice**. As a result, many parts of speech have paradigms, i.e. sets of all possible word forms, ranging from eight to ten forms for nouns to more than 100 for verbs including participle forms. Despite the vast amount of forms, many of them are created in very regular ways.

The traditional way of presenting Latin morphology is by using tables for each inflection class. With the tools provided by GF, a similar approach is possible. To give an example, we can have a look at nouns: Nouns in Latin are inflected by **case** and **number** and have an inherent **gender**. For the noun *villa* the full paradigm can be seen in Figure 6.1. It is ***feminine*** and follows the first declension class, as can be seen in the lexical information above the table in the figure. All forms are given in a two-dimensional table, one dimension representing the **number** and the other the **case**. Exactly the same information contained in Figure 6.1 can be encoded in GF using tables and records (Figure 6.2). The result is a record with two fields, `Form` and `Gender`, one containing the paradigm and one containing the inherent **gender**.

Based on this encoding, we want to define a general pattern that creates all correct word forms for a given lemma. To do this, we can define a similar table, but mapping from **case** to the suffix that has to be attached to the stem to give the correct form (Figure 6.3). Here `lemma` is a placeholder for an arbitrary noun lemma.

To use this kind of inflection schema in our lexicon we can use the name `declension1lemma` as a function, i.e. as a placeholder for the whole construct in Figure 6.3. To be able to use it, we have to get the stem from the lemma

$$
\texttt{villa\_N} = \begin{bmatrix}
\texttt{Form} & \begin{bmatrix}
\textit{singular} \Rightarrow & \begin{bmatrix}
\textit{nominative} & \Rightarrow \textit{vīlla} \\
\textit{genitive} & \Rightarrow \textit{vīllae} \\
\textit{dative} & \Rightarrow \textit{vīllae} \\
\textit{accusative} & \Rightarrow \textit{vīllam} \\
\textit{ablative} & \Rightarrow \textit{vīllā} \\
\textit{vocative} & \Rightarrow \textit{vīlla}
\end{bmatrix} \\
\textit{plural} \Rightarrow & \begin{bmatrix}
\textit{nominative} & \Rightarrow \textit{vīllae} \\
\textit{genitive} & \Rightarrow \textit{vīllārum} \\
\textit{dative} & \Rightarrow \textit{vīllīs} \\
\textit{accusative} & \Rightarrow \textit{vīllās} \\
\textit{ablative} & \Rightarrow \textit{vīllīs} \\
\textit{vocative} & \Rightarrow \textit{vīllae}
\end{bmatrix}
\end{bmatrix} \\
\texttt{Gender} & \textit{female}
\end{bmatrix}
$$

Figure 6.2: GF paradigm for *vīlla*

$$
\texttt{declension1(lemma) = get stem from lemma and use in}
\begin{bmatrix}
\texttt{Form} & \begin{bmatrix}
\textit{singular} \Rightarrow & \begin{bmatrix}
\textit{nominative} & \Rightarrow \texttt{stem}+a \\
\textit{genitive} & \Rightarrow \texttt{stem}+ae \\
\textit{dative} & \Rightarrow \texttt{stem}+ae \\
\textit{accusative} & \Rightarrow \texttt{stem}+am \\
\textit{ablative} & \Rightarrow \texttt{stem}+\bar{a} \\
\textit{vocative} & \Rightarrow \texttt{stem}+a
\end{bmatrix} \\
\textit{plural} \Rightarrow & \begin{bmatrix}
\textit{nominative} & \Rightarrow \texttt{stem}+ae \\
\textit{genitive} & \Rightarrow \texttt{stem}+\bar{a}rum \\
\textit{dative} & \Rightarrow \texttt{stem}+\bar{\imath}s \\
\textit{accusative} & \Rightarrow \texttt{stem}+\bar{a}s \\
\textit{ablative} & \Rightarrow \texttt{stem}+\bar{\imath}s \\
\textit{vocative} & \Rightarrow \texttt{stem}+ae
\end{bmatrix}
\end{bmatrix} \\
\texttt{Gender} & \textit{female}
\end{bmatrix}
$$

Figure 6.3: Generalization of the first declension class for nouns. The + operator attaches a suffix directly to a string

contained in the lexicon. This part is straightforward for regularly inflected words. For example, for many nouns it is sufficient to split off the nominative singular suffix. If we apply `declension1` to the lemma *vīlla*, it will be converted to the stem *vīll* by removing the suffix, and we get the same result as in Figure 6.2, i.e. `villa_N = declension1(`*vīlla*`)`. Here we assume ***feminine* gender** for nouns of the first declension. This assumption is valid for the majority of nouns in this inflection class. For all nouns in this class that have a different inherent gender, such as *agricola*, which is ***masculine***, we can explicitly give this diverging information in the lexicon. However, in the majority of the cases, the inferred gender is correct and does not have to be given explicitly. This reduces the information that has to be given in the lexicon.

```
smartnoun(lemma) = get suffix from lemma and use in
```

`suffix` is *-a*	⇒ `declension1(lemma)`	*Comment: e.g., terra, gender feminine*
`suffix` is *-us*	⇒ `declension2(lemma)`	*Comment: e.g., hortus, gender masculine*
`suffix` is *-um*	⇒ `declension2(lemma)`	*Comment: e.g., verbum, gender neuter*
`suffix` is *-er*	⇒ `declension2(lemma)`	*Comment: e.g., ager, gender masculine*
`suffix` is *-ū*	⇒ `declension4(lemma)`	*Comment: e.g., cornū, gender neuter*
`suffix` is *-ēs*	⇒ `declension5(lemma)`	*Comment: e.g., rēs, gender feminine*

Figure 6.4: Example of a smart paradigm for the noun inflection classes. Nouns of the fourth declension, e.g., *cāsus* cannot be handled here because they conflict with the second case, e.g., *hortus*

In the end we have many similar functions, one for each inflection class. To select the correct function, we can use some heuristics which automatically selects the appropriate function based on the lemma. This kind of heuristics is called a smart paradigm (Détrez and Ranta, 2012). An example for a smart paradigm for nouns is shown in Figure 6.4. For regular words the lemma is sufficient to unambiguously determine the inflection class and based on it, the whole paradigm as well as the default gender. For irregular words or words that diverge from the assumed standard features, additional information is required, for example for nouns the ***genitive singular*** form as well as the **gender**. The use of smart paradigms allows us to use compact lexica, usually only consisting of a small number of word forms for each lexicon entry.

6.2.3 Syntax

The Latin syntax is often considered very flexible and allows for free word order. This is facilitated by the fact that a lot of the syntactic information is already encoded in the morphological forms. Instead of requiring a fixed sequence of constituents, agreement of morphological forms can be used to define the phrase structure.

Latin allows for multiple levels of free word order phenomena. The most obvious one is the reordering of top-level constituents. This means that usually there is a predominant word order, e.g., Subject-Object-Verb, for each author or period, basically any other combination is considered grammatical as well. The

literature mentions limitations on the word order, but they usually are motivated by semantics and pragmatics (Pinkster, 1984, chap. 9.3.2.2). However, our focus is purely on syntax, so we have to be able to model all orders. Some of these word orders are rarely used, but not impossible, as Bamman and Crane (2006) show in an analysis of Latin treebanks.

It is possible to implement flexible word order using records and tables. Records can be used to store parts of a phrase, and tables, together with complementary features, can be used to decide the order on a higher level.

$$\texttt{exampleSentence} = \begin{bmatrix} \texttt{Subject} & [\text{“imperator”}] \\ \texttt{Verb} & [\text{“tenet”}] \\ \texttt{Object} & [\text{“imperium”}] \end{bmatrix}$$

$$\texttt{combineSentence(sentence)} = \begin{bmatrix} \boldsymbol{SVO} => & [\texttt{Subject ++ Verb ++ Object}] \\ \boldsymbol{VSO} => & [\texttt{Verb ++ Subject ++ Object}] \\ \boldsymbol{VOS} => & [\texttt{Verb ++ Object ++ Subject}] \\ \boldsymbol{OSV} => & [\texttt{Object ++ Subject ++ Verb}] \\ \boldsymbol{OVS} => & [\texttt{Object ++ Verb ++ Subject}] \\ \boldsymbol{SOV} => & [\texttt{Subject ++ Object ++ Verb}] \end{bmatrix}$$

$$\texttt{combineSentence(exampleSentence)} = \begin{bmatrix} \boldsymbol{SVO} => & [\textit{imperator tenet imperium}] \\ \boldsymbol{VSO} => & [\textit{tenet imperator imperium}] \\ \boldsymbol{VOS} => & [\textit{tenet imperium imperator}] \\ \boldsymbol{OSV} => & [\textit{imperium imperator tenet}] \\ \boldsymbol{OVS} => & [\textit{imperium tenet imperator}] \\ \boldsymbol{SOV} => & [\textit{imperator imperium tenet}] \end{bmatrix}$$

Figure 6.5: Function combining parts of a sentence depending on a parameter controlling constituent order. The `++` operator combines words into a sequence.

To just handle the top-level word order we can create a new parameter controlling for it and use it in a table to organize the constituents we have in separate record fields (Figure 6.5). This makes it easy to encode all possible word orders for three constituents, e.g., with the `exampleSentence` we can form *imperator imperium tenet* and *imperator tenet imperium*, but also *imperium tenet imperator*. We also immediately handle the special cases $\boldsymbol{SV}, \boldsymbol{VS}, \boldsymbol{VO}$ and $\boldsymbol{OV}$, because both the `Subject` and `Object` field can be empty, e.g., when the subject pronoun is dropped or the direct object is only implied. This is of course a very simplified view on Latin word order, but this is also just a starting point.

Another free word order phenomenon is the interleaving of phrases, called hyperbaton. A phrase can be made discontinuous by inserting other words of the same sentence they are part of. In general, discontinuous phrases can be created using records with record fields for each part of the discontinuous

phrase. For example, a sentence consists not just of a subject, verb and object as constituents, but it is further separated into additional parts, which can be placed independently. This includes for example splitting the subject noun phrase into the noun phrase without any potential determiner or quantifier and the separate determiner. The additional parts on the sentence level, and where they can be placed, are shown in Table 6.1. The order of these parts is controlled using additional parameters that allow exactly the positions named in the table.

As a result, in the current version of the grammar, six parameters control the word order, one of which is the top-level word order described before. Besides that, the position of adjectives, verb-modifying adverbs, sentence adverbials and the sentence complement, e.g., the direct object, can be determined by independent parameters . These parameters already allow for a broader range of word orders than just the top-level orders. Being able to separate parts of the noun phrase and having a parameter that decides that the verb can be put between the subject noun and the determiner, allows handling the hyperbaton in the beginning of the Bellum Gallicum: *Gallia est omnis [...]* (Caes. Gall. 1, 1, 1). In Section 6.3.2.2 we will have a closer look at the sentence starting with this phrase.

However, it is still just an approximation of all word order phenomena possible. To handle hyperbaton in a more general way, it has been suggested to extend the grammar formalism with a general interleave operator (Ljunglöf, 2004, Section 5.4). Until it is included in the canonical GF, interleaving has to be implemented using discontinuous constituents and tables to control the word order.

Constituent	Placement
Verb	Verb in regular position Subject noun phrase interleaved with verb
Sentence adverb	Sentence adverb before subject Sentence adverb before verb sentence adverb before object Sentence adverb before negation particle
Subject determiner	Determiner before subject noun Determiner after subject noun
Verb complement	Verb complement before the verb Verb complement after the verb

Table 6.1: Additional parts of a sentence and their placements

Reordering words in a sentence is only part of the solution to Latin syntax. Another important aspect is agreement between words in various parts of a sentence. This agreement can even span long distances. Agreement in our Latin grammar can be enforced by using the right features, stored in record fields, to select the correct word forms from tables. This way of handling agreement is similar to passing features through the syntax tree from the point where they are defined to the place where the information is needed. One

example is the the object case or preposition used with a transitive verb, which is an inherent feature of the verb. This information is passed on until the verb is combined with the direct object where it enforces the correct noun form. Another example is the noun gender, inherent to the noun, that is passed on in a specific record field until it is used to determine the gender of the adjective forms in the noun phrase.

6.2.4 Lexicon

The RGL itself only includes a very limited lexicon consisting of 350 lemmata, based on the Swadesh list (Swadesh, 1952; Swadesh, 1955) consisting of 207 primitive concepts, with some more modern concepts added (Ranta, 2011, p. 233). To provide translations for the modern concepts into Latin, the Latin Wikipedia (Vicipaedia, 2018) proved a useful resource.

However, for most applications additional lexical resources are necessary. We created a large lexicon based on Whitacker's Words (Whitaker, 2006) which contains 39,225 entries of which 37,404 were automatically converted into a format suitable to be integrated into the Latin GF grammar. The lexical information provided by Whitacker's Words includes, for each entry, one or many base forms, part of speech, as well as information about the inflection. Most of this information is used in the GF lexicon to improve the result when generating the full forms. Adding this lexicon broadens the potential applications of the grammar by broadening the available vocabulary and only with the extended lexicon the evaluation in the following section was possible.

6.3 Evaluation

One important aspect of this paper is the assessment of the quality of the Latin grammar. Even though it already proved to be a useful resource for a language learning application (Lange and Ljunglöf, 2018a; Lange, 2018), the grammar has never been evaluated on authentic language data.

We designed four experiments to fill this gap. The experiments are designed to give direct results about the quality of the grammar, but they are also general prototypes of how to evaluate similar grammars. We evaluate morphology and syntax under both quantitative as and qualitative aspects, in order to get a clear idea of the coverage, and at the same time to provide a more in-depth analysis of the reasons that lead to errors.

For the first three experiments, we use two corpora: the complete PROIEL treebank as well as the parts of Caesar's *Commentarii de Bello Gallico* included in the Universal Dependency treebank, which itself is a subset of the PROIEL treebank. We report the results for both corpora separately and attempt an error analysis on the smaller Caesar corpus. We release all data and code used in the evaluation to the public, which allows everyone interested to reproduce the results or adjust and reuse the methodology for their own needs.[2]

[2] `https://doi.org/10.17605/OSF.IO/UWJ59` (Lange, 2020)

6.3.1 Morphology

We present two methods to test our description of Latin morphology. The first approach tests what ratio of Latin lemmata and word forms occurring in the corpus can be analyzed by the Latin grammar. To provide means of comparison we also show the results for two other morphological analyzers for Latin.

To test the quality of the analyses, in the second part, we compare the results of our morphological analyzers to the morphological annotation in the treebank. We assume that this mostly manual annotation is correct and can be used as a gold standard. For both approaches we provide an error analysis.

6.3.1.1 Experiment 1: Morphological Coverage

We want to evaluate our own implementation of Latin morphology on both the smaller Caesar corpus and the complete PROIEL treebank. To put the results of our morphological analyzers into context, we compare them to both LEMLAT (Passarotti, Budassi, et al., 2017) and LatMor (Springmann, Schmid, and Najock, 2016).

LatMor uses finite state technology for the morphological analysis, which allows it to both analyze word forms as well as generate word forms from a lemma and a list of features. LEMLAT uses lists of morphemes and rules how to combine them to analyze word forms. LEMLAT's approach is limited to analysis and does not provide means of generation. Both LEMLAT and LatMor are purely morphological analyzers for Latin and exist as standalone applications. Our morphological analyzer uses an approach called functional morphology (Forsberg and Ranta, 2004) and provides both means of analysis and generation. More importantly, it is one of the components of a larger grammar system.

		Caesar		PROIEL	
		Lemmata	Wordforms	Lemmata	Wordforms
Caesar		2,162	5,308	8,519	29,560
LEMLAT	Analyzed	2,127 (98%)	5,260 (99%)	8,189 (96%)	29,066 (98%)
	Missing	34	48	330	494
LatMor	Analyzed	2,108 (97%)	5,012 (94%)	7,944 (92%)	26,599 (89%)
		54	296	575	2467
GF	Analyzed	1,956 (90%)	4,460 (84%)	6,884 (80%)	23,771 (80%)
	Missing	206	848	1,635	5,789

Table 6.2: Result of the coverage in the first experiment

As a first step, we extract from the corpora both a list of all lemmata and of the word forms that occur. We then separately analyze the forms in the lists with the three tools. The results can be seen in Table 6.2. LEMLAT is able to analyze the most word forms and the numbers for lemmata and word forms are close to each other. For LatMor the results are quite similar concerning lemmata, but substantially more word forms are missing. Our system covers 90% of the lemmata in Caesar and 80% of the lemmata in the

Lemmata		Word forms	
Total missing lemmata	206	Total missing word forms	848
Lemmata missing in lexicon	134	Word form missing due to missing lemma	357
Lemmata missing due to error in grammar	47	Wrong word form	491
Spelling variations	24		
Annotation errors	22		

Table 6.3: Error analysis for the first experiment on the Caesar corpus

whole of PROIEL. In both cases, about four times the amount of word forms is missing, as compared to the number of missing lemmata.

A closer analysis of the missing lemmata shows various reasons why the analysis failed (see Table 6.3). In certain cases, several reasons can occur at the same time. As a consequence, the sum for the subcategories for lemmata is larger than the number of all errors. The most common reason is items missing from the lexicon. This can be explained by comparing the lexical resources used by the three tools. LEMLAT uses a lexicon compiled from three lexical resources with 40,014, 26,250 and 82,556 entries respectively. LemLat includes a lexicon based on an updated version of the Berlin Latin Lexicon, consisting of ca. 70,000 entries. Our lexicon, based on Whitacker's Words is the smallest of the three with 37,404 entries. Other reasons involve spelling variations like *i/j* and annotation errors in the treebank, where for a word form a wrong lemma is given.

If we only focus on errors caused by our grammar and ignore all 357 word forms that are missing as a consequence of missing lemmata, we end up with a list of 47 lemmata and 491 word forms. These numbers are comparable to the results of LatMor. The remaining errors can directly guide efforts for improving the morphology in the grammar.

6.3.1.2 Experiment 2: Quality of the Morphological Analysis

The second experiment assesses the quality of the analyses produced by the Latin grammar. To do this, we compare the output of our morphological analyzers with the morphological features that are included in the treebank.

Universal Dependencies uses a universal set of features[3], which is intended to provide a uniform way to describe various languages. GF has some uniform features defined by the RGL, but other GF features can be very specific, both depending on the language and the author of the grammar. To test the quality of the morphological analysis we need to be able to translate between UD and GF features.

Although some of the features are easy to map because they are directly equivalent to features in the other formalism, other translations are more difficult, e.g., because one formalism is more fine-grained than the other. One such example is parts of speech. While Universal Dependencies uses just the

[3]https://universaldependencies.org/u/feat/index.html

label `VERB` for all verbs, independent of the verb valency, GF uses categories depending on valency, e.g., `V` for intransitive verbs, `V2` for transitive verbs, `VQ` for question-complement verbs and `VS` for sentence-complement verbs. All these variants of verbs have to be mapped to the same representation.

Features	Values
POS	Adj, Adv, N, Num, PN, V, Prep, Det, Pron, Conj, Interj
Case	Nom, Gen, Dat, Acc, Abl, Voc
Number	Sg, Pl
Gender	Masc, Fem, Neutr
Degree	Positive, Comparative, Superlative
Voice	Active, Passive
Mood	Indicative, Subjunctive, Imperative
Tense	Present, Imperfect, Perfect, Future, PluPerfect, FuturePerfect, Past
Person	Person1, Person2, Person3
Reflexivity	Reflexive, NonReflexive
VerbForm	Infinitive, Finite, Participle, Supine, Gerundive, Gerund
PronType	Relative, Personal, Possessive, Interrogative, Reciprocal
Polarity	Negative

Table 6.4: Common features and values used to compare GF analyses and UD annotation

To test the quality of our analysis, for each token in the corpus, its analysis and the annotation in UD have to be translated into a common representation. The list of all features and values used in this common representation can be seen in Table 6.4. Each analysis is represented as a list of features, and for each feature there is a list of values. The word forms of the tokens can be lexically or morphologically ambiguous, which leads to several candidate analyses, both using GF and within UD.

	Total word forms	Matched	Missing
Caesar	4,460	3,957 (88%)	503
PROIEL	23,771	20,380 (86%)	3,391

Table 6.5: Results of the coverage in the second experiment

For each token we acknowledge that the analysis matches the annotation if one of the candidates provided by GF matches a candidate provided by UD. The numeric result of the matching can be seen in Table 6.5. For Caesar 88% of the morphological analyses match with the Universal Dependencies annotations and for the whole of PROIEL 86% of analyses match. As a result, for 88% of all tokens, GF provides an analysis which matches one of the annotations used in UD.

We conducted a closer analysis of the cases where we failed to match analyses with annotation in the Caesar corpus. It shows various reasons for mismatches (Table 6.6). The errors can be caused by problematic annotations in the treebank, or in the implementation of the grammar, or by missing information in the lexicon. It can even have several reasons at the same time,

	UD	GF	Lexicon
Errors	228	206	138
Part of Speech	91	52	137
Case	4	18	1
Tense	4	12	-
Deponent verbs	77	0	-
Numerals	17	3	-
Participles	3	34	2
Pronouns	77		-

Table 6.6: Detailed error analysis for the second experiment on the Caesar corpus

such as errors both in the grammar and the lexicon. Again, the sum of the errors in the subcategories can be larger than the total number of errors.

The most common problem is when the parts of speech do not match. For example, when a word form can be both analyzed as a noun and an adjective, then it can happen that the gold standard requires the analysis as an adjective but the grammar analyses it as a noun. This can also be caused by missing information in the lexicon. Sometimes only one of the two lexical items, the noun or the adjective, is in the lexicon.

Sometimes there is a mismatch in other grammatical features like **case**, **number** or **tense**. **Case** mismatches are most common in the grammar, especially in connection with gerund forms. Also, problems with **tense** occur in the grammar, usually in cases where a verb form can be analyzed both as ***present*** and ***perfect*** **tense**. In these cases, the grammar fails to provide one of the two analyses.

In some cases, the UD annotations cause problems, e.g., all deponent verbs are annotated as ***passive*** forms while the grammar analyses them as ***active*** forms. Also, numerals are often not annotated as such, but instead as adjectives. There can be different motivations for different annotation styles. This can be handled in annotation guidelines. When these guidelines are missing, it is up to the annotator to decide on the annotation. Concerning deponent verbs, the morphological forms match passive verb forms, so annotation as passive voice can be justified and in UD the syntactic function follows the morphological form.[4] However, some kind of annotation to denote deponent verbs would be useful as well.

Especially pronouns and participles suffer from potential inconsistencies in the annotation. Participles tend to be annotated either as adjectives or as verb forms. There often exist separate dictionary entries for verbs and their participles and the annotation in the treebank can vary between the two choices. More problematic, however, are pronouns, under which we also subsume adjectives and adverbs with pronominal function. They can be annotated and analyzed in many different ways, and with the current method we cannot properly compare their features between GF and UD. For that

[4]Discussion of the issue: https://github.com/UniversalDependencies/docs/issues/713. Accessed June 29th, 2020

reason, we do not distinguish error cases for UD and GF for pronouns and only report 77 cases where problems occurred when matching pronoun analyses.

6.3.2 Syntax

To test the quality of the syntax rules is more challenging than to test the morphology morphology. We tested the morphology in a context-independent manner while for syntax we have to take larger context into account.

The strength of GF as a grammatical framework is, among other things, its flexibility. GF grammars can be used both to analyze and to generate natural language data. Also, the grammars can easily be adapted for specific use cases. UD on the other hand is mostly used to train models for specific natural language processing tasks. In itself it can neither be used to analyze nor to generate new text. Besides this limitation, UD provides a valuable resource for linguistic research, based on its corpus data.

General syntactic analysis is a challenge for GF because natural language in an unrestricted setting tends to be very ambiguous, which leads to a large number of analyses. This makes syntactic analysis computationally expensive; the generative capabilities are not impacted as much. To solve this issue, a combination of the features of GF and UD can be used to test GF grammars, circumventing the challenges of unrestricted syntactic analysis.

6.3.2.1 Experiment 3: Coverage of Syntactic Constructions

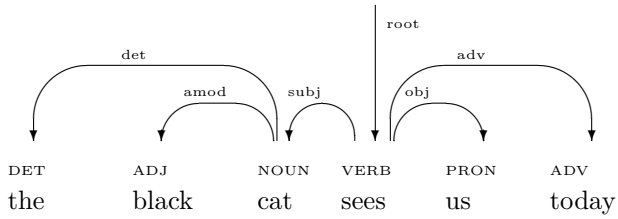
The core of the GF resource grammar library contains 357 functions[5] plus language and application-specific extensions. Out of the 357 core functions, 284 functions are included in the Latin grammar and 73 have not been implemented yet.

In the context of an evaluation the relevant questions are: which of these rules are actually used in the Latin corpus, which rules are the most commonly used and how many of them are not covered by the Latin grammar.

To answer these questions, we can translate the UD dependency trees found in the treebank into GF phrase structure trees. Such a translation method was presented by Ranta and Kolachina (Ranta and Kolachina, 2017; Kolachina, 2019). It is based on the assumption that syntax rules of a phrase structure grammar can be recovered from the dependency label and the parts of speech in the dependency tree. An example can be seen in Figure 6.6. To achieve this, several challenges have to be overcome.

- Dependencies usually are binary but GF rules can combine more than two constituents
- GF has a finer-grained distinction of lexical categories than the parts of speech used by Universal Dependencies (Figure 6.6b). As a consequence, POS tags have to be expanded to all possible lexical categories, which can lead to a combinatorial explosion

[5]In the context of GF, the terms (syntactic) functions and (grammar) rules can be used interchangeably



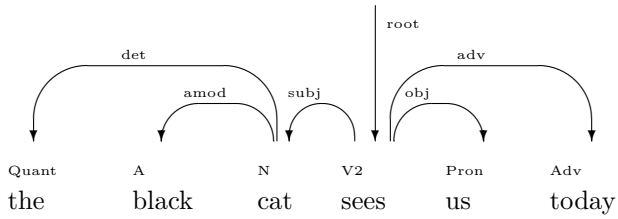
(a) The original dependency tree

UD	GF
DET	Det
ADJ	A,AP,...
NOUN	CN,N,...
VERB	V,V2,...
PRON	Pron
ADV	Adv,AdV,...

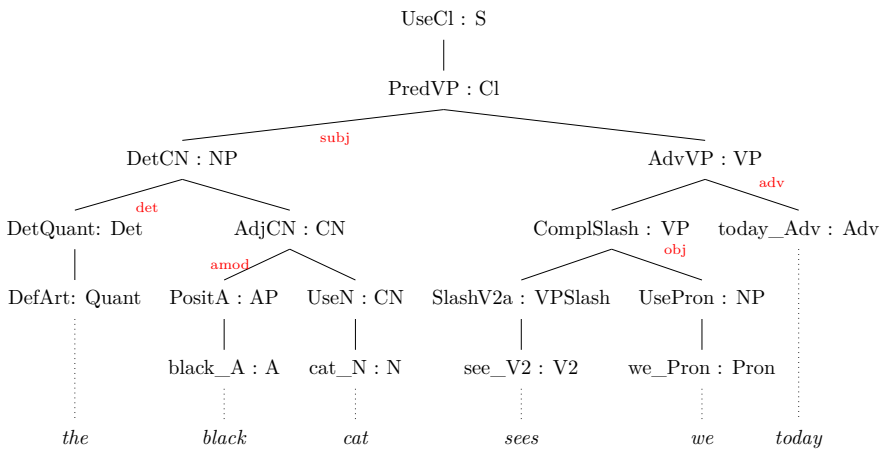
(b) Mapping between UD POS tags and GF categories

Name	Label
AdjCN	amod head
AdVVP	adv head
DetCN	det head
ComplSlash	head obj
PredVP	subj head
...	

(c) UD labels for GF abstract syntax functions



(d) The dependency tree with correct GF categories



(e) The abstract syntax tree annotated with UD labels

Figure 6.6: Example for the connection between UD trees and GF abstract syntax trees (Kolachina, 2019, p. 11)

- We don't want to fail completely when we have problems translating a subtree. To recover from an error, generic "Backup" rules, that construct subtrees of a certain category even when they are not covered by the grammar, can be used.

We converted both the complete PROIEL treebank and the Caesar fragment from dependency trees into GF abstract syntax trees. Based on these trees and the assumption about revovering phrase-structure rules from dependency trees, we extracted the list of RGL functions that are used in the trees. When we compare these lists with the list of all RGL functions implemented in our grammar, we get the results shown in Table 6.7.

	Caesar	PROIEL
Occurrences syntactic functions in all trees	79,361	779,630
Occurrences Backup functions in all trees	36,988	338,730
Occurrences RGL functions in all trees	36,050	361,607
Occurrences other functions in all trees	6,323	79,293
Occurrences Latin RGL functions in all trees	35,221	353,031
Occurrences missing Latin RGL Functions in all trees	829	8,576
Number of distinct syntactic functions in all trees	249	318
Number of distinct Backup and other functions in all trees	118	156
Number of distinct RGL functions in all trees	131	162
Number of distinct Latin RGL functions	108	132
Number of distinct missing Latin RGL functions	23	30

Table 6.7: Results of the third experiment: number of abstract and "Backup" functions occurring in the treebanks

Converting all trees of the Caesar corpus into GF trees results in trees containing in total 79,361 syntactic functions, out of which 47% are "Backup" functions and 45% are RGL functions. The remaining 8% of functions are not part of the core RGL. When looking only at distinct functions, we encounter 249 different ones, out of which 131 are part of the RGL. The whole PROIEL treebank offers similar numbers.

We see that 23 of the constructions not implemented in the Latin grammar are used in the treebank. In the whole PROIEL treebank the number of unimplemented rules increases to 30. However, these rules make up only a small fraction of all rule occurrences, around 0.1% in both corpora. Also, when ranking the RGL rules that can be found in the corpora, none of the missing functions is among the top 20 rules.

A relevant issue is the high number of "Backup" nodes. To investigate why the "Backup" nodes occur and how they can be avoided remains for future work.

6.3.2.2 Experiment 4: Generative Expressivity of the Syntax

Finally, we want to show the generative expressivity of our grammar. To do so, we selected a sentence and show how it can be constructed based on our Latin grammar. The sentence we selected is the first sentence of the Bellum Gallicum (Caes. Gall., 1, 1, 1):

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam, qui ipsorum lingua Celtae, nostra Galli appellantur.[6]

This sentence provides several challenges for a syntactic analysis. The verb *est* is placed in the middle of the subject noun phrase, thus forming a hyperbaton. It contains two relative clauses, one specifying the three parts of the country and one specifying the name of the people living in the third part. In several parts we find ellipsis where either the subject or the verb is not mentioned explicitly. However, agreement has to be guaranteed, even in cases where words are left out and between nouns and pronouns, e.g., between *lingua* and *nostra*. Finally, we can see various high-level word orders including Subject-Verb and Verb-Subject.

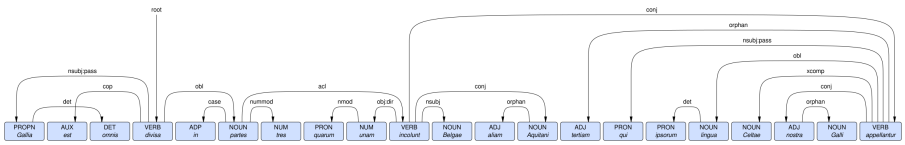


Figure 6.7: Dependency tree for Caes. Gall., 1, 1, 1

For comparison the dependency structure from the UD treebank can be seen in Figure 6.7. In contrast, we created the phrase structure tree describing this sentence based on the Latin resource grammar. The complete syntax tree for the sentence is presented in Figure 6.8. This tree, for example, clearly shows the discontinuous subject noun phrase *Gallia omnis* with the verb *est* in the middle.

Some of the complexities in the tree are due to the structure of the RGL and some language specific extensions to the core RGL are necessary to cover all aspects of the sentence. However, the main point of this experiment is to show that it is feasible to use our grammar to generate authentic sentences, similar to sentences that can be found in a real-world corpus.

[6]Translation following McDevitte & Bohn (Caesar, 1869): *All Gaul is divided into three parts, one of which the Belgae inhabit, the Aquitani another, those who in their own language are called Celts, in our Gauls, the third.*

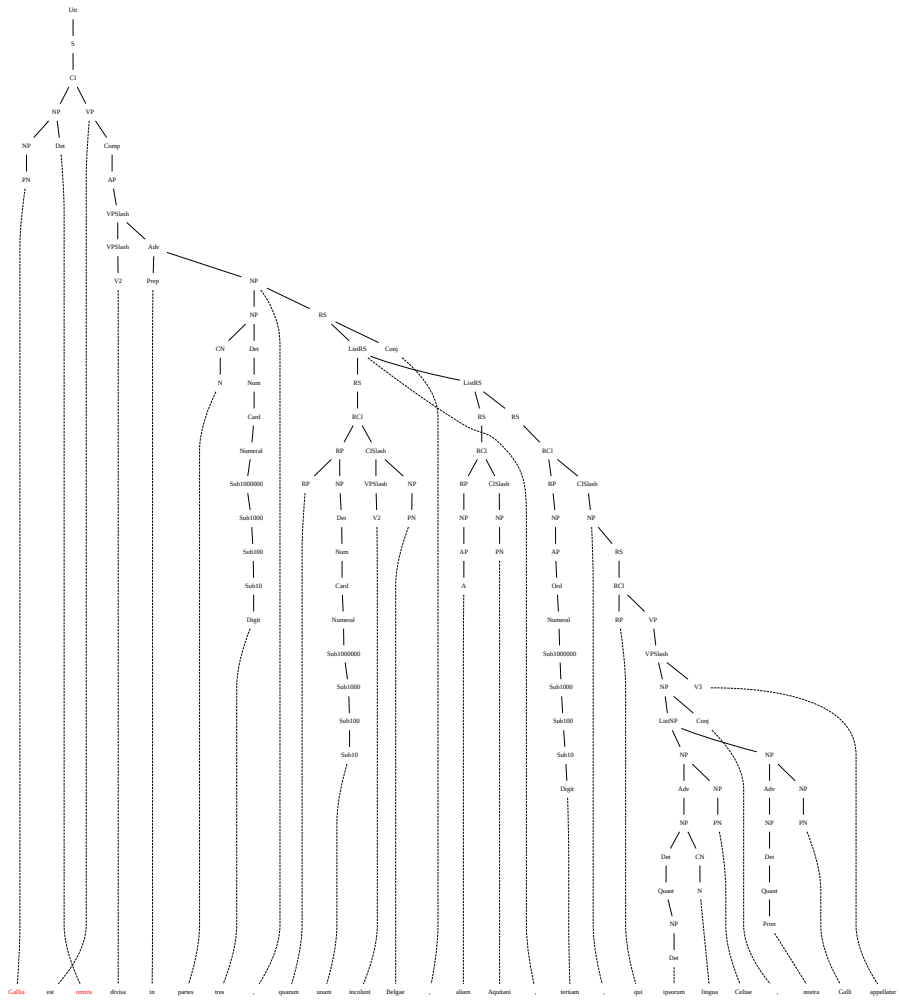


Figure 6.8: Parse Tree for Caes. Gall., 1, 1, 1

6.4 Conclusion

This article gives an overview over an open source computational Latin grammar. The intended use is to provide a linguistically motivated resource for a wide range of computational applications. Besides the use in applications, it can also be used to test linguistic hypotheses. The strength of this grammar, besides syntactic analysis, is in language generation. One applications that makes use of our grammar and its strength is a language learning application for Latin (Lange and Ljunglöf, 2018a; Lange, 2018). Other applications are imaginable, e.g., translation systems based on GF's inherent multilinguality as well as various systems to provide access to cultural heritage. These other applications remain, however, for future work.

Furthermore, we present four experiments to evaluate the quality of this Latin grammar. The results of this evaluation are relevant to show both the current quality of the grammar and to provide a benchmark which can be used to measure improvements. This is helpful for future efforts in developing this grammar.

Besides these direct results, this article describes methods to evaluate GF resource grammars in general. This has never been done before in any comparable way. The main reason for the lack of evaluation of resource grammars is the challenge of syntactic analysis in unrestricted settings, suffering from a high degree of ambiguity. Only with the methods to translate between UD and GF trees (Kolachina and Ranta, 2016; Ranta and Kolachina, 2017; Kolachina, 2019) it became possible to evaluate syntactic quality on a large scale.

Future work includes the use of the results from our evaluation to improve the grammar and fix the issues uncovered by the experiments. For example, the first experiment identified errors in our formalization of the third verb inflection class. The second experiment showed problems in various places, including problems with verb tenses and gerund forms. The third experiment shows which of the missing functions are most relevant to improve syntactic coverage. The results of the experiments can server as a point of reference for future improvements.

Another option to solve the problems with our morphology would be to integrate an external morphological analyzer. Both LEMLAT and LatMor are analyzers with a high level of accuracy. Using these external resources would be possible but would, at the same time, require some restructuring of the grammar, making it less self-contained. The idea to use external morphological analyzers in GF grammars has come up before, but remains for future considerations for our Latin grammar.

Part III

Grammar-Based CALL

Chapter 7

Paper III: MULLE: A Grammar-Based Latin Language Learning Tool to Supplement the Classroom Setting

Herbert Lange and Peter Ljunglöf

Published in , *Proceedings of the 5th Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA '18),*
Melbourne, Australia, 2018

Abstract

MULLE is a tool for language learning that focuses on teaching Latin as a foreign language. It is aimed for easy integration into the traditional classroom setting and syllabus, which makes it distinct from other language learning tools that provide standalone learning experience. It uses grammar-based lessons and embraces methods of gamification to improve the learner motivation. The main type of exercise provided by our application is to practice translation, but it is also possible to shift the focus to vocabulary or morphology training.

7.1 Introduction

Computer-assisted language learning (CALL) is a growing field that is also more and more in the focus of the general public thanks to popular tools such as Duolingo[1] or Rosetta Stone.[2] In combination with the rise of the smartphone it has become possible to learn languages almost any time and anywhere in an entertaining way.

Text input on mobile devices equiped with touch screens as the primary input device can be difficult, but is relevant to language learning tasks. This general problem led to the development of several alternative input methods (Ward, Blackwell, and Mackay, 2002; Kumar, Paek, and Lee, 2012; Felzer, MacKenzie, and Rinderknecht, 2014; Shibata et al., 2016) including Ljunglöf's method of grammar-backed word-based text editing (Ljunglöf, 2011).

We present the MUSTE[3] Langueage Learning Environment (MULLE)[4], an application for language learning that combines several techniques: tree-based sentence modification, controlled natural language grammars for the exercise creation as well as concepts of gamification.

The goal of our system is to provide a tool that enriches the traditional language learning setting in an enjoyable way and helps to avoid problems with learner motivation that can be encountered in language classes.

7.2 Previous and related work

MULLE is based on an underlying theory of word-based grammatical text editing by Ljunglöf Ljunglöf (2011).

The software used to translate between the surface text and the syntax trees is the Grammatical Framework (GF) (Ranta, 2009a; Ranta, 2011). It is a grammar formalism and parsing framework based on type theory. On top of this formalism, a multilingual library of grammars is build, the so-called resource grammar library (RGL) (Ranta, 2009b) which covers more than 30 languages including Latin (Lange, 2017). It provides an interface that can be used to implement more application-specific grammars similar to an application programming interface (API) in computer programming.

An important aspect of CALL is the factor of both long and short-term motivation for which the concept of gamification is relevant (Deterding et al., 2011). Several approaches are possible, of which we focus on GameFlow by Sweeter and Wyeth Sweetser and Wyeth (2005) and MICE by Lafourcade (described in Fort, Guillaume, and Chastant (2014, section 4)). GameFlow translates the more general Flow approach (Csikszentmihalyi, 1990) to computer games.

Finally, comparison to other language systems is relevant for our work. Most language systems share common features, especially translation exercises seem quite similar across different systems. Still there are major differences in

[1] `https://www.duolingo.com/`
[2] `http://www.rosettastone.eu/`
[3] `http://www.cse.chalmers.se/~peb/muste.html`
[4] `https://github.com/MUSTE-Project/MULLE`

the way these systems work and the use cases they are developed for. Duolingo for example heavily relies on text input created by the user, uses a mixture of user-generated content and machine learning techniques (Kenji Horie, 2017) and is meant for open independent online learning mostly for modern languages. MULLE on the other hand uses resources created by experts, does not require text input created by the user, and is intended for, but not restricted to, accompanying language classes in a closed classroom setting.

7.3 Creation of interactive exercises from a Latin textbook

The idea of grammar-based text modification led us to the creation of MULLE. It is game-like and the player solves language learning exercises focusing on translation. Each exercise consists of two sentences in different languages, one language that the user already knows (i.e. the metalanguage), and the language to be learned (i.e. the object language). Both sentences differ in some respect, depending on the grammatical features that the lesson is focusing on.

Using GF together with the RGL helps us to create domain-specific grammars in a straightforward way. Such grammars can be designed to catch exactly the complexity of the lessons in a classic textbook. That way we can mirror the same lesson structure in MULLE, at the same time adding more flexibility and giving the possibility of generating a large supply of interactive exercises with plenty of variation using vocabulary and concepts familiar from class and textbook.

A textbook for language learning is usually split into a sequence of lessons with texts and exercises of growing syntactical complexity. This is the case for textbooks both at high-school and university levels (e.g. Lindauer, Westphalen, and Kreiler (2000) and Ehrling (2015)). Typically, each chapter consists of a text part, a vocabulary list, some grammatical explanation and additional exercises. The growing vocabulary and increase in complexity helps the student learn the whole of a language in a slow pace. This approach is also common in language learning applications and can readily be implemented in MULLE.

Each grammar lesson in MULLE covers a set of interactive exercises. So we need lesson-specific grammars that use the same lexicon and grammatical constructions as the corresponding parts of the textbook. For that we can use the RGL, when writing a new grammar for a lesson we already have access to an extensive description of the languages we want to cover and only have to select the concepts we want to include.

First a lexicon is created that covers exactly the vocabulary of a lesson. Extensive lexical resources are already available for GF and they can easily be extended by the author of the grammar relying on the morphological component of the grammar to generate the correct word forms.

Next the grammatical constructions that will be used in this lesson are selected by exposing only the parts that are relevant to the planned learning outcomes. The RGL can be seen as a collection of grammatical constructions, and each lesson uses a subset of these concepts. So by only providing a

restricted subset together with the selected vocabulary it is possible control the complexity of the lessons.

Finally every grammar we create needs to be multilingual for at least two languages: the metalanguage (e.g. Swedish), and the object language (e.g. Latin). Since the RGL is inherently multilingual it is straightforward to provide the lessons in multiple languages; With only minimal adjustments we can cover as many languages as we want as long as they are already included in the RGL.

The usual size of the lesson grammars we encountered so far was between 50 and 100 lexical items and about 20 syntax rules.

The main focus of our work is on one form of translation exercises but other forms of exercises are also useful in the context of language learning. That usually includes explicit vocabulary exercises and, in the case of languages with a strong morphology like Latin, some exercise for practicing word forms.

Practicing vocabulary is possible either by using lexical categories as top-level categories of the syntax trees or by using sentences that are almost correct except for a lexical mismatch in one position.

Exercises for morphology involve slightly more work since our grammar formalism by default only creates grammatical sentences including correct word agreement. So to be able to practice morphology in our setup have to relax these morphological constraints in the grammars. That gives us a way to create exercises where the user has to both identify wrong morphological forms in a sentence and find the right form to replace them with.

7.4 Implementation

Based on these ideas we have implemented MULLE which can already be used in language classes. In order to be independent of certain kinds of devices and operating systems we provide the whole application as a browser-based online application.

The application is developed independent of the grammars that can be used. That means that the whole system can be set up by providing the application with a set of lesson grammars and a fully usable language learning environment is available.

7.4.1 User interface

The user interface is kept minimalist, as can be seen in Figure 7.1, and only provides the user with the most essential information, including the current score count, the sample sentence in the metalanguage and the modifiable sentence in the object language that has to be altered to match the sample, and the time elapsed since starting the exercise as well as clicks spent on the exercise.

Colours are an important aspect of the interface because they indicate progress in solving the exercise. The background colours of the words highlight which parts of the two sentences already match up with each other. In the example *kejsaren* is a proper translation of *Caesar* which is shown by highlighting them in the same colour. The same is the case for both occurrences of

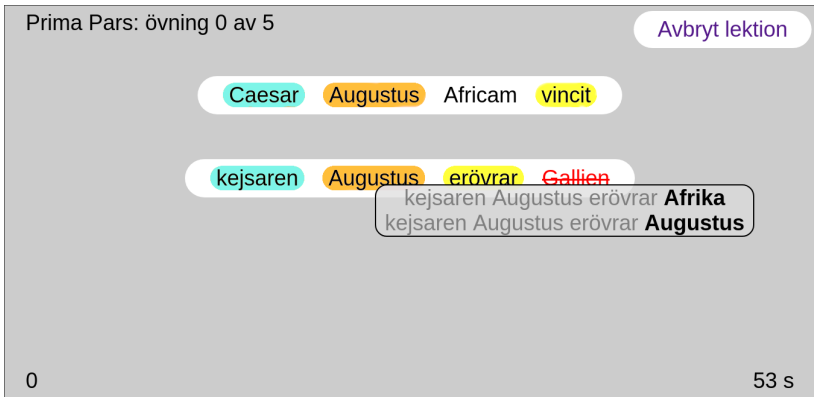


Figure 7.1: Screenshot of the exercise view

Augustus as well as the pair *vincit* and *erövrar*. The meaning of the colours is that phrases in the same colour are translations of each other. Only one pair of words, *Africam* and *Gallien*, is not highlighted, so here some user intervention is needed.

This current design reduces the possible distractions while supporting the learner. Depending on the target age group a more elaborate graphics design could have a more positive effect on the acceptance of the system.

7.4.2 Gamification

We presented two approaches for gamification in Section 7.2, based on which we selected certain aspects to be included in our application. For our application the following features of GameFlow seem most relevant: *Concentration*, i.e., minimising the distraction from the task, *Challenge* by giving a scoring schema, *Control* by providing an intuitive way to modify the sentence, *Clear goals* by providing a lesson structure, and *Immediate feedback* with the colour schema.

The concept of lessons and exercises is essential for this kind of language learning because it makes the learning progress explicit. The completed lessons are presented to the student together with the scores, so that they can see their own progress on the way to reaching their final goal of learning the language.

By applying methods from GameFlow, we positively influence the motivation while learning a new language. Adding more features of gamification, especially involving social aspects, is a possible extension for the future.

7.5 User interaction

After logging into the system the user is presented with a list of lessons and the current status, i.e. the number of finished exercises for each lesson and the current score. Some lessons might be disabled because they require previous lessons to be completed first. Now the user can choose one of the enabled lessons to start the exercises.

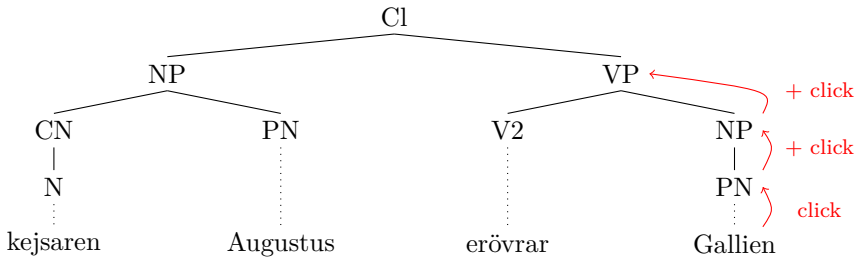


Figure 7.2: Syntax tree including the path through the tree after several clicks on the word *Gallien*

As soon as a user starts a lesson a set of exercises is selected. These exercises are chosen from a list of exercises in a database. The exercises consist of two syntax trees that different in certain grammatical aspects. Associated with each syntax tree is one sentence, one in the metalanguage and one in the object language. The syntax trees are hidden from the user and only implicitly influence the user experience.

The exercises are presented in the form shown in Figure 7.1. The background colours of the words show the state of the translation. When the user clicks on one of the words in the bottom sentence, they are presented with a list of potential replacements. This selection is based on where in the tree the word is introduced. In the example the user clicked on the word *Gallien*, which is a proper name, so all proper names contained in the grammar are presented. By clicking several times on the same word the focus can be expanded to cover larger phrases, e.g. from proper name to noun phrase, and so on, by traversing upwards through the tree (Figure 7.2). The menu contains all phrases of the syntactic category selected by clicking on words. That means that suggestions can contain more or less words than currently in focus. So for example if a noun phrase is in focus, both noun phrases with and without adjectives appear in the list. Selecting a longer phrase is the same as inserting words in the sentence and selecting shorter phrases corresponds to deleting words from the phrase.

With these operations, i.e. substitution, insertion, and deletion, the user can modify the sentence to finish their task. When the two sentences are proper translation of each other, i.e. the two syntax trees are similar, the user is congratulated on the success and presented the final score.

Lessons can be interrupted and resumed at any time as well as repeated to improve the score.

7.6 Evaluation

For the evaluation of our approach we have designed an experiment setup. The full setup includes a basic placement test in the beginning that is repeated at the end of the test period to provide information about the learning outcome. The placement test consists of a fixed set of exercises from all lessons that will

be covered during the experiment period. Both error rate and completion time are measured. A questionnaire controls for factors like learner background, previous knowledge, etc. It also gives insight into the learner motivation in the beginning so it can be repeated in the end to see any development in this relevant aspect. Then over the span of the experiment the students can use the software independently online. The lessons are kept in sync with the syllabus of the course that is accompanied by the experiment. In the end the collected data consists of changes in learning outcome and learner motivation as well as activity of the student in the system.

In a pilot experiment we tried aspects of this experimental evaluation. The results were not yet statistical significant because the course size was very small and the dropout rate was high. From the initial 10 Students only 4 finished the course so we only received complete feedback from two students out of initially 6 participants. Anyways, the general interest, both by teachers and students, in this kind of application is strong.

A larger scale follow-up experiment will focus on the change in the learner attitude, which is relevant for showing that our tool is suited for tackling potential anxiety in learners, a problem Latin teachers have pointed out (Dimitrijevic, 2017). With more participants different kinds of control and test conditions can be introduced.

7.7 Discussion

One challenge with the user interface is the semantics of clicks, especially concerning word insertion. Clicking on a gap between two words to insert words seems more intuitive than clicking on a word. But where to click might also depend on the languages involved.

Another important question for the current application is the influence of the grammar design both on the learning experience and the learning outcome. It is possible to vary the design of the grammar to change the behaviour of our system.

Related is the role of semantics in the lesson grammars. The lessons and exercises are meant for learning the syntax of a language but nonsensical semantics can be an obstacle for the learning process. For example the famous sentence *Colorless green ideas sleep furiously* (Chomsky, 1957, p. 15) is considered grammatical but would probably distract the learner.

7.8 Future work

This project is work in progress and we plan to extend the system in several ways. First, we will repeat the experiment from Section 7.6 on a larger scale. Furthermore we plan to extend our implementation to become more feature-rich with a special focus on investigating the points addressed in the discussion section. Finally we want to continue collaborating both with teachers and students to improve the system in order to enrich teaching and learning Latin.

Chapter 8

Paper IV: Putting Control into Language Learning

Herbert Lange and Peter Ljunglöf

Published in, *Proceedings of the 6th International Workshop on Controlled Natural Languages (CNL 2018), Maynooth, Ireland, 2018*

Abstract

Controlled Natural Languages (CNLs) have many applications including document authoring, automatic reasoning on texts and reliable machine translation, but their application is not limited to these areas. We explore a new application area of CNLs, the use of CNLs in computer-assisted language learning. In this paper we present a web application for language learning using CNLs as well as a detailed description of the properties of the family of CNLs it uses.

8.1 Introduction

Controlled Natural Languages (CNLs) are an active field of research in computational linguistics. Their definition usually is vague, but the general consensus is that they are constructed languages placed somewhere between full natural languages on the one hand and formal languages on the other. Kuhn discusses the definition in (Kuhn, 2014) and presents typical applications for CNLs, such as machine translation and document authoring. He also expects other applications without going into detail about these possibilities. In this paper we present one new application of CNLs, the use in computer-assisted language learning (CALL).

We present the MUSTE language learning environment (MULLE)[1], a tool for learning languages as a second or foreign language which can use CNLs for the description of learning objectives, and automatically generate translation exercises from formal grammars of these languages. The application itself uses methods similar to conceptual authoring (Hallett, Scott, and Power, 2007) to edit sentences in natural languages in order to make them proper translations of each other. By using CNL grammars as the basis for the exercises, and using textbook lessons as the basis of the grammars, it is possible to create a learning environment that complements the traditional classroom setting.

This article is structured as follows: In Section 8.2 we present related work, both in CNLs and CALL. Section 8.3 describes the language learning application we designed. It describes the interface provided to the user and gives details about the grammars used. In Section 8.4 we sketch an experimental method to evaluate our system and describe a small-scale pilot. Finally in Section 8.5 we discuss further questions and conclude the article in Section 8.6 with a look to possible future work.

8.2 Related Work

The use of CNLs for language learning seems to be a new application in this field that has not yet been broadly discussed. For this reason, the amount of directly related work is rather limited with the exception of (Abolahrar, 2011). However, this application can also be viewed as a combination of two tasks that have been popular among the CNL community: reliable machine translation and user support for text edition and creation.

8.2.1 Related CNL work

Quite often the motivation for the use of Controlled Natural Language is their proximity to formal language. This allows, e.g. automatic reasoning within and reliable translation of these languages. To guarantee that the language used by an author is covered by a CNL, special editors have been proposed. Two approaches are conceptual editing (Hallett, Scott, and Power, 2007) and predictive editing (Kuhn and Schwitter, 2008; Angelov and Ranta, 2010).

[1] `https://github.com/MUSTE-Project/MULLE`

Predictive editing uses chart parsers and compatible grammars to suggest only valid continuations in the process of writing. Some systems only support a fixed vocabulary while other systems support the extension of the vocabulary while the document is authored.

Conceptual editing instead refines the underlying representation by manipulating the surface presentation, i.e. the natural language sentence. In this process so called "holes" are created and filled.

Angelov and Ranta (Angelov and Ranta, 2010) not only present a predictive editor, but also suggest a translation system based on Attempto Controlled English (ACE) (Fuchs et al., 2005). It provides reliable machine translation via abstract syntax trees in the Grammatical Framework (GF) (Ranta, 2009a; Ranta, 2011).

Some CNLs were designed to aid learning languages at a time before computers were considered a tool for it. One example is Basic English (Ogden, 1930), an auxiliary language created to help people learn English as a second or foreign language, invented at the beginning of the 20th century.

8.2.2 Related work within language learning

In the field of language learning applications several approaches can be observed. They range from finite-state technology for morphology training (Kaya and Eryiğit, 2015) over annotated text data, or semantic resources combined with rule-based algorithms (Moritz et al., 2016; Michaud, 2008; Redkar et al., 2017) to user-generated content in combination with machine learning (Kenji Horie, 2017). The aim and scope of theses systems also varies broadly, including the learning environment they target. The systems (Michaud, 2008; Redkar et al., 2017) target a closed classroom setting with specific language classes while (Kaya and Eryiğit, 2015; Moritz et al., 2016) aim at a broader learning environment and are applicable outside a specific course. Modern general-purpose systems like Duolingo[2] target independent language learners.

Reliability also varies between these systems. The smallest scale systems provide the most reliable examples while the most general systems being the least reliable.

8.3 Application: Language Learning using CNLs

In this paper we describe a web-based language learning tool which takes advantage of CNL grammar features. It is intended for use in a closed, classroom-related learning environment and provides reliable translation exercises by using fully formalised grammars. The tool presents exercises grouped into lessons where the user is presented with sentences in two different languages.

Usually two languages are involved in language learning: one language the user already knows is used for instructions in the language classes (the meta language) and one language the user is learning (the object language) by

[2] `https://www.duolingo.com`

discussing it in the meta language . So the meta language and object language in the classroom determine the two languages used in our exercises.

The task of the user is to edit one of the sentences to make it a proper translation of the other. Currently, this means that the underlying GF abstract syntax trees for both sentences have to be the same. As a future development, we imagine an extension where we consider not single trees but sets of trees, in order to be able to handle ambiguous parses. In this case it would be sufficient to have at least one abstract tree in the intersection of the two sets.

The tool has been used in an introductory course in Latin for Swedish students. So in all examples given here Swedish acts as the meta language and Latin as the object language. We implemented the first four lessons of (Ehrling, 2015) and conducted the pilot of a user study that is described in more detail in Section 8.4.

8.3.1 The editing interface

Our application uses a method for word-based text-editing (Ljunglöf, 2011), which is in principle related to conceptual editing. It uses syntax trees as formal representations and provides editing operations like insertion, deletion, and substitution on the surface by mapping them onto tree operations. In our application we only look at complete syntax trees, that means we do not use "holes" for incremental creation, but instead modify complete syntax trees.

The editing operations work in the following way: the user clicks on a word in the sentence or on the gap between two words in the sentence. The click position is translated to the node in the syntax tree in which the word is introduced or the closest node covering the space that was clicked. Based on the subtree below this node, all subtrees with the same root category are computed and their linearisations, i.e. their surface representations, are collected and presented to the user.

To clarify this process a set of screenshots with the corresponding syntax tree can be seen in Figures 8.1a–8.1e. In the screenshot one can see the two sentences in different languages. The sentence at the top is fixed and the sentence at the bottom can be changed by the user. The syntax tree beside the screenshot describes the sentence at the bottom. Figure 8.1a shows the start of the system before any click. Clicks on words in the sentence are then translated to pointers into the tree. For example clicking on the word "Gallien" will set the pointer (circled node) to the node introducing this string, in this case the rightmost PN node (Figure 8.1b). Then the category in the focus of the pointer is used to compute all similar trees of with the same category in the root. These trees are used to suggest potential replacements which are shown to the user in the form of a menu. Clicking on the same word several times moves the pointer up in the tree which changes the root category of the candidate trees from PN to NP, VP and so on, and suggests different changes to the sentence (Figure 8.1c–8.1d) before finishing in the root of the tree with category Cl (Figure 8.1e) where the menu does not change anymore. In this way larger phrases can be changed at the same time of more global features like sentence negation can be modified. When instead the user clicks on a different

position than before, then the system is reset and the pointer again points to the node indicated by the new click.

8.3.2 Lessons and exercises

The application provides of a set of lessons, each consisting of a number of exercises. A lesson is defined by a multilingual GF grammar which is derived from a part of a textbook. These parts usually consist of text fragment (a sample can be seen in Figure 8.2), a vocabulary list, some explanation of grammatical phenomena, as well as some exercises that are supposed to be solved on paper.

We adopt this structure in our system by using the same text fragments presented in the textbook lessons and formalising them in separate grammars that cover the vocabulary as well as the syntactic constructions used in the corresponding parts of the textbook.

Given a lesson grammar, an exercise consists of two syntax trees and their surface representations. With the task described before a score is calculated based on both the number of clicks and the time spent before finishing the exercise. After finishing a certain number of exercises, the lesson is considered finished.

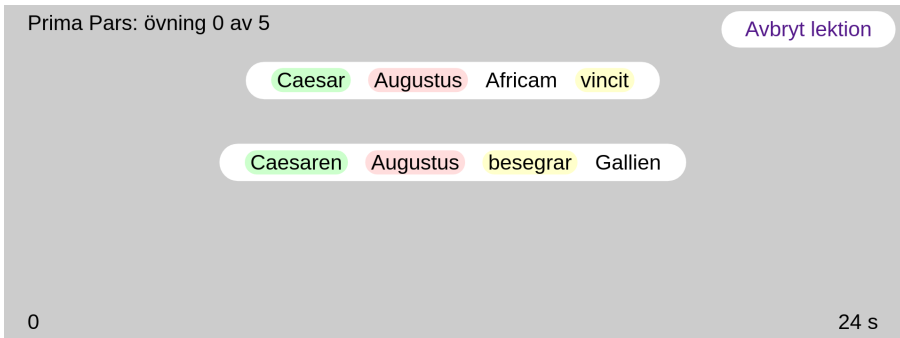
8.3.3 Creating the lesson grammars

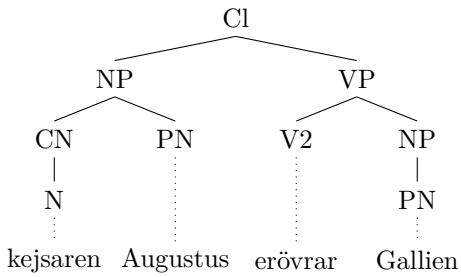
We create a GF grammar for each textbook lesson in the following way. This work should be automated as much as possible but the basic procedure can also be executed manually.

1. The first step is to adapt a lexicon for the textbook lesson, which is given as an explicit vocabulary list in the book. We can use existing reliable lexical resources or GF smart paradigms (Détrez and Ranta, 2012) to implement it in our grammar.

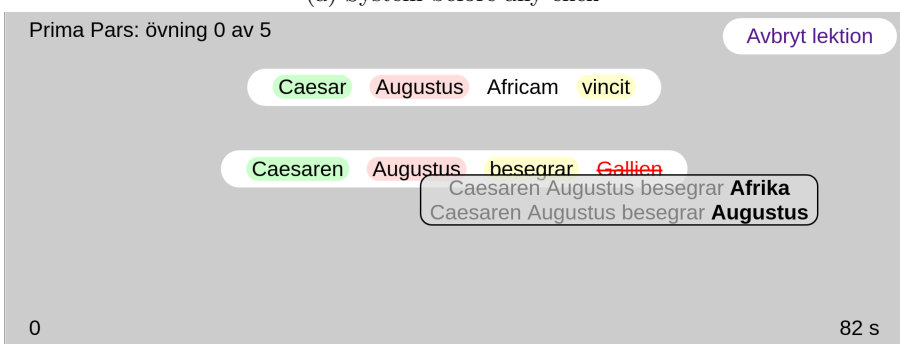
2. The next step is to create syntax trees for all sentences in the text (Figures 8.3a–8.3b). This can either be done manually or semi-automatically. To automate this process, we parse each sentence using the GF resource grammar library (RGL) (Ranta, 2009b) augmented with the lexicon from step 1. Because of syntactic ambiguities this might result in several possible trees, so afterwards we have to manually select the correct syntax tree, i.e. the desired analysis of the sentence. This involves some linguistic knowledge from the person creating the grammar.

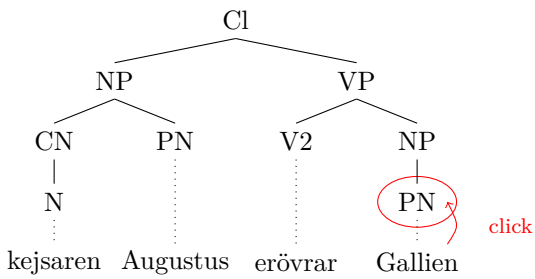
3. Finally, we formulate the grammar that describes the text fragment in the textbook. This can be done straightforwardly by reading off the grammar rules from the internal nodes in the trees (see Figure 8.3c). This will usually result in an over-generating grammar, so we use different techniques to reduce the over-generation such as merging two or more generated rules into one.



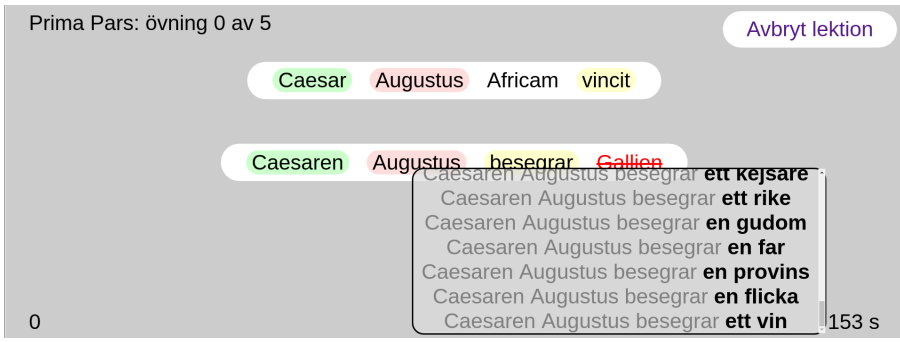


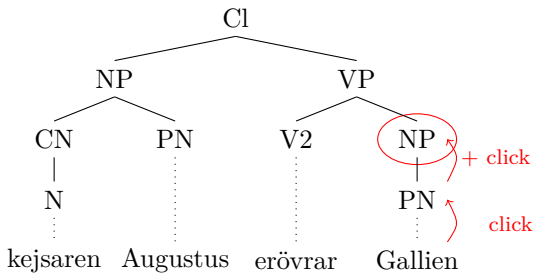
(a) System before any click



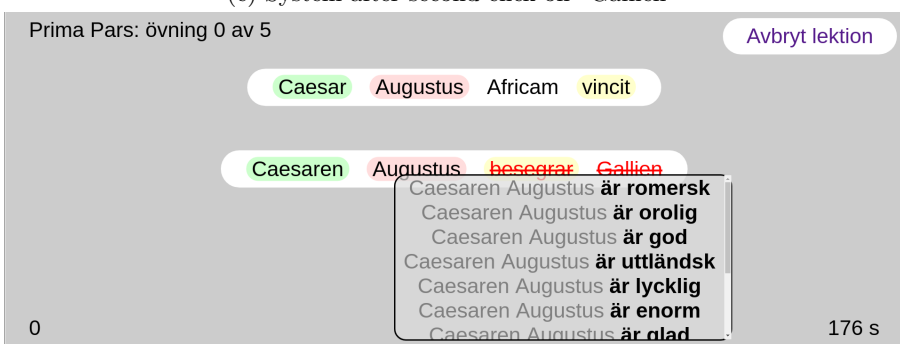


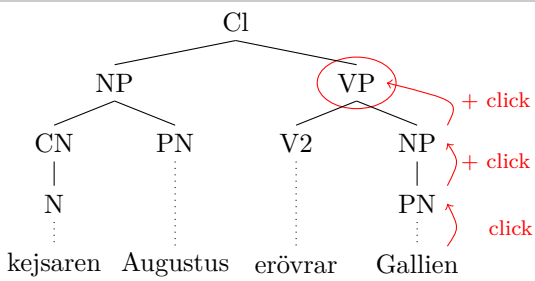
(b) System after one click on “Gallien”



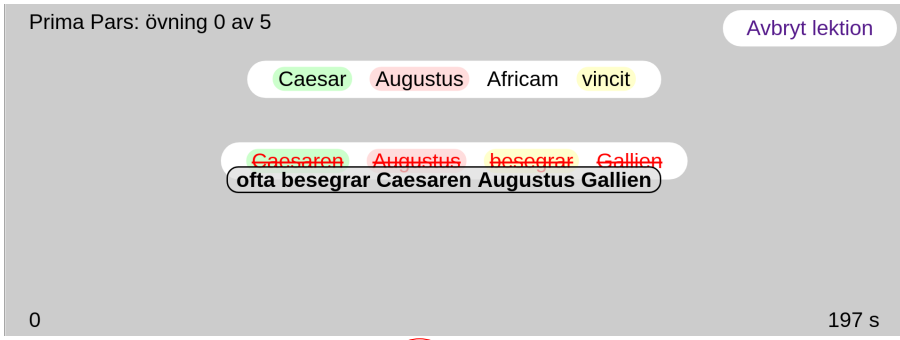


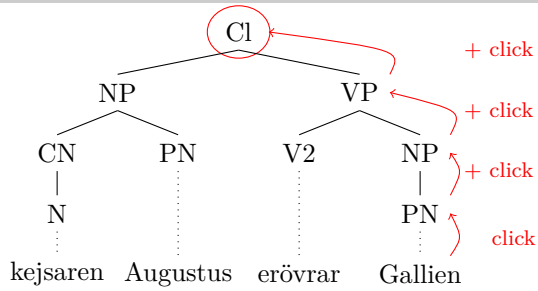
(c) System after second click on “Gallien”





(d) System after third click on “Gallien”





(e) System after fourth click on “Gallien”

```
Prima scripta Latina

[...] Imperium imperatorem habet. Imperator imperium tenet.
Caesar Augustus imperator Romanus est. Imperium Romanum tenet.
Multas civitates externas vincit. Saepe civitates victae
provinciae  deveniunt.
[...]
```

Figure 8.2: Sample from the text fragment in the first lesson in (Ehrling, 2015)

8.3.4 Making the lesson grammars ungrammatical

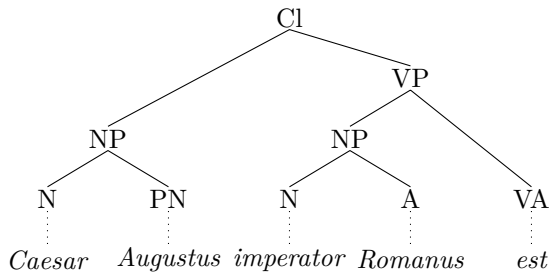
The above process yields a grammar which only accepts syntactically correct sentences. However, we also wish to train the morphology in the object language, e.g. noun-verb agreement, number/gender inflection, affixation, etc.

It is possible to semi-automatically transform a lesson grammar into a grammar that accepts some grammatical errors, e.g. sentences where nouns and verbs disagree in number. What has to be done manually is to indicate which inflection parameter(s) in which grammar rule(s) should be loosened. Then it is possible to automatically transform the grammar into a grammar that accepts sentences where that specific inflection parameter is violated.

The user’s task still is to edit a sentence in the object language to make it a translation of the meta language, but now they will have the additional complexity of allowing ungrammatical sentences. It is possible to control the

```
Caesar Augustus imperator Romanus est.
```

(a) Example sentence



(b) Syntax tree derived from sentence in Figure 8.3a

```
NP   -->  N PN
NP   -->  A N
VP   -->  VA NP
Cl   -->  NP VP
S    -->  Cl
...
```

(c) Derived abstract grammar

level of ungrammaticality by deciding how many inflection parameters should be loosened.

8.3.5 Characterisation of the lesson grammars

We identified relevant criteria that characterise grammars that are suitable to be used by our application. The two most relevant criteria are a layer of semantics in the grammars as well as making some implicit features of the syntax explicit.

Grammars in GF usually are distinguished between resource grammars and application grammars. The main difference is that resource grammars just describe the syntax of a language without any semantic considerations while application grammars are used for a specific application and include semantic aspects necessary for that domain.

The grammars that are used in our system also have a strong focus on syntax, but require at least some semantic restrictions. From a purely syntactic point of view, adjectives can be combined with any noun, but language use only allows certain combinations. This can be solved by including semantic knowledge in different ways, of which the inclusion of FrameNet-style semantics in the grammars (Gruzitis and Dannélls, 2017) seems the most natural.

The second point is, that natural languages might employ syntactic features that are not visible on the surface. One example are romance languages that allow the dropping of pronouns in the subject position because the relevant information is already present in the verb form. But to keep the grammars multilingual, these pronouns still have to be present in the grammar as empty

tokens, which is an obstacle in language learning. This has to be changed in a way to make this information explicit on the surface for our application to be useful.

8.3.5.1 PENS classification

Given this description of the family of grammars we designed we want to render this definition more precisely in the PENS classification scheme (Kuhn, 2014). PENS stands for Precision, Expressiveness, Naturalness, and Simplicity and is typically used to classify CNLs. Each of these scales ranges from 1 to 5 with 1 being the least and 5 the highest level of strictness possible.

Our grammars are fully specified in a computational grammar formalism and each sentence can be mapped to a set of abstract syntax trees. We do not insist on completely unambiguous interpretations but the set of interpretations will always be finite. According to the PENS classification that places our languages in the field of *Deterministically interpretable languages* (P^4).

The sentences generated by our grammars should be syntactically correct and in this aspect be considered as *Languages with natural sentences* (N^4). Because we focus just on sentences in our application, an extension to *Languages with natural texts* is not necessary.

The scope of our grammars is very limited, both by the text fragments and the explicit vocabulary, which makes it possible to formalise the language fragments in compact grammars. Even though they rely on external resources in the form of the RGL, the fragments can be considered as *Languages with short descriptions* (S^4).

The only problematic dimension is Expressivity, because we do not really focus on a translation to a specific logic interpretation, but remain on the level of the abstract syntax tree and its expressiveness. But because this is not relevant for our application we decide to ignore this dimension and set it to E^- That places our languages in the family of $P^4E^-N^4S^4$ languages.

This classification is not just some characterisation of the grammars we use in our system now and the languages defined by them, but instead a general requirement for all grammars and languages that can be used in our framework.

8.4 Evaluation

To test the acceptance of our application, as well as the desired change in learning outcome and learner motivation, we designed an empirical evaluation which we partially conducted as a pilot in connection with an introductory course for Latin at university level.

The full experiment consists of a prepared set of four lessons with a runtime of about four weeks. At the beginning, the students are asked to answer a questionnaire to control for aspects of the learner background and give some insight into the motivation at the beginning of the course. A simple timed placement test with eight exercises, four from each lesson, estimates the language skills before taking the class. The participants then are split into one treatment group and one or several control groups. In the following four

weeks the students in the treatment group get access to lessons matching the progress in class while one control group only gets access to the traditional learning material in the text book. After the experiment period the students are given a slightly modified version of the questionnaire from the beginning to test for a change in motivation and a second placement test to see if there is a change in speed to solve the exercises.

In the pilot, due to lack of students, we could only ask for general feedback without gaining relevant insight into change in learner motivation or learning outcome. From ten students in class six volunteered to try the application and answer the first questionnaire. But due to a general drop out from this class, only four students were present in the end, of which only two had volunteered to participate. Still, the general feedback from both teachers and students was very positive which encourages us to aim for a full scale version of the evaluation.

8.5 Discussion

In the related work we pointed out several different technical approaches for systems in the field of foreign and second language learning. The different systems differ not only in the expressivity of the underlying technology but also in their intended use case.

Systems which employ technology with limited expressivity like finite-state technology aim at a closed setting in a very specific classroom setting but provide a high reliability. Other systems that employ very expressive machine learning methods can be used in a very open and classroom-independent setup but suffer from a lack of reliability.

With our system, which uses a very expressive syntax formalism, we currently target a closed classroom setting where we can profit from the reliability of our grammar-based approach but we also believe it is possible to widen the focus to provide a completely open language-learning application.

We claim that our system employs controlled natural languages for language learning. Some might disagree, and we admit that the PENS classification fails in the point of expressivity. But the application we sketch is grammar-agnostic, which means one can use almost any multilingual grammar to generate translation exercises from it. The grammars we used so far might not really be seen as controlled languages because they are defined too implicitly, even though textbook lessons usually are created with clear concepts in mind. Still it can be used with any grammar that fulfils the PENS requirements we identified as characteristic for our grammars. This also gives a chance for further research looking into the application of CNLs in CALL far beyond the scope of this work.

Finally it is possible to discuss the combination of the underlying technology with other CNLs to build different applications. We think that there is some potential, especially given the similarity of conceptual editing and the word-based text editing, to have a fruitful exchange between the CNL community and other disciplines.

8.6 Conclusions and Future Work

We presented a working application usable for language learning that uses fully formalised grammars to define language learning lesson. According to some definition of controlled natural languages these lessons can be seen as CNLs, even there might be problems with this claim.

In the future we want to investigate how the design of the grammars influences the learning experience. This mostly concerns the structure of the grammars with varying focus on syntax and semantics. But that also includes additional ideas for different kinds of language learning exercises.

Another relevant topic of research is automatic generation of "good" exercises. This in entangled with the questions which kind of exercises besides translation exercises we want to include in our application. It also seems connected to a different topic, the selection of good examples in the creation of lexica (Kilgarriff et al., 2008), even though features of good translation exercises are not exactly the same as for good lexicon examples. Still this would give an opportunity for further interdisciplinary research.

Part IV

Learning Domain-Specific Grammars

Chapter 9

Paper V: Learning Domain-Specific Grammars From a Small Number of Examples

Herbert Lange and Peter Ljunglöf

Submitted to , *Special Issue: Natural Language Processing in Artificial Intelligence - NLPinAI 2020, in: Series "Studies in Computational Intelligence" (SCI), Springer*

Abstract

In this chapter we investigate the problem of grammar learning from a perspective that diverges from previous approaches. These prevailing approaches to learning grammars usually attempt to infer a grammar directly from example corpora without any additional information. This either requires a large training set or suffers from bad accuracy. We instead view learning grammars as a problem of grammar restriction or subgrammar extraction. We start from a large-scale grammar (called a *resource grammar*) and a small number of example sentences, and find a subgrammar that still covers all the examples. To accomplish this, we formulate the problem as a constraint satisfaction problem, and use a constraint solver to find the optimal grammar. We created experiments with English, Finnish, German, Swedish and Spanish, which show that 10–20 examples are often sufficient to learn an interesting grammar for a specific application. We also present two extensions to this basic method: we include negative examples and allow rules to be merged. The resulting grammars can more precisely cover specific linguistic phenomena. Our method, together with the extensions, can be used to provide a grammar learning system for specific applications. This system is easy-to-use, human-centric and can be used by non-syntacticians. Based on this grammar learning method we can build applications for computer-assisted language learning and interlingual communication, which rely heavily on the knowledge of language and domain experts who often lack the competence to develop required grammars themselves.

9.1 Introduction

Many currently common trends in natural language processing (NLP) are aiming at general purpose language processing for tasks such as information retrieval, machine translation and text summarization. But there are use cases where it is not necessary to handle language in general. Instead, it is possible to restrict the language which such a system needs to recognize or produce. The reason for restricting the language can be in requirements of very high precision, e.g., in safety-critical systems, or in order to build domain-specific systems, e.g., special-purpose dialogue systems. In this context, domain-specific means that an application is used in a specific application domain, and is unrelated to the concept from cognitive science.

One common aspect of these high-precision applications is that they can be built using computational grammars to provide the required reliability and interpretability. These domain-specific grammars often have to be developed by grammar engineers who are not always experts in the application domain. Domain experts on the other hand usually lack the skills necessary to create the grammars themselves.

We present a method to bridge the gap between the two parties. Starting from a general-purpose *resource grammar*, we use example sentences to automatically infer a domain-specific grammar. We apply constraint satisfaction methods to ensure that the examples are covered. It is also possible to apply various objective functions to guarantee that the result is optimal in a certain way.

This chapter is an extension of our paper from the Special Session on Natural Language Processing in Artificial Intelligence at the 12$^{\text{th}}$ International Conference on Agents and Artificial Intelligence (ICAART 2020) (Lange and Ljunglöf, 2020). In addition to the technique presented in the original paper, in this chapter we extend the basic method in two ways. Firstly, we account for negative examples. Besides sentences that have to be included in the language of the new grammar, it is also possible to give sentences which the grammar should not be able to parse. This allows for an iterative refinement process. Secondly, we generalize the method, which uses grammar rules as the basic units, to subtrees of size[1] larger than 1, as the basic units. In combination with the ability to merge rules, we move away from extracting pure subgrammars, towards learning modified grammars. The resulting grammars can more precisely cover specific linguistic phenomena.

For our experiments we use the Grammatical Framework (GF) (Ranta, 2009a; Ranta, 2011) as the underlying grammar formalism, but the main ideas are transferable to other formalisms such as head-driven phrase structure grammar (HPSG) (Pollard, 1994), lexical-functional grammar (LFG) (Kaplan and Bresnan, 1982; Bresnan, 2001) or lexicalized tree-adjoining grammar (LTAG) (Joshi and Schabes, 1997). The main requirements are that there is a general purpose resource grammar, such as GF's resource grammar library (RGL) (Ranta, 2009b), and that parse trees can be translated into logic constraints in a way similar to what we present in this chapter.

[1]The size of a tree is the number of nodes in the tree

9.1.1 Use Case: Language Learning

Our primary use case is language learning. We have developed a grammar-based language learning tool for computer-assisted language learning (Lange, 2018; Lange and Ljunglöf, 2018b; Lange and Ljunglöf, 2018a), which automatically generates translation exercises based on multilingual computational grammars.

Each exercise topic is defined by a specialized grammar that can recognize and generate examples that showcase a topic. Creating those grammars requires experience in grammar writing and knowledge of the grammar formalism. However, language teachers usually lack these skills.

Our idea is to let the teacher write down a set of example sentences that show which kind of language fragment they have in mind, and the system will automatically infer a suitable grammar. One exercise topic could focus on gender and number agreement, another one on relative clauses, while yet another could focus on inflecting adjectives or the handling of adverbs.

The optimal final grammar should cover and generalize from the given examples. At the same time it should not over-generate, i.e., cover ungrammatical expressions, and instead reduce the syntactic ambiguity, i.e., the number of syntactic analyses or parse trees, as much as possible. Completely covering the examples, generalize from the examples and not being over-generating usually are mutually exclusive and contradictory requirements. So the best we can hope for is a balance between these requirements.

9.1.2 Use Case: Interlingual Communication

Another use case for our research is domain-specific applications, such as dialogue systems, expert systems and apps to support communication in situations where participants do not share a common language. These situations can, for example, be found in the healthcare sector, especially when involving immigrants (Ranta, Angelov, et al., 2017). Here misunderstandings can cause serious problems.

In the development of such systems, the computational linguists who are specialists on the technological side have to collaborate with informants who have deep knowledge of the language and domain. A common domain is established by discussing example sentences. These sentences can be automatically translated into a domain-specific grammar, which can be refined by generating new example sentences based on the initial grammar and receiving feedback about them from the informants.

Such an iterative, example-based, development of application-specific grammars allows for close collaboration between the parties involved. The result is a high quality domain-specific application.

9.2 Background

We want to give some insight into the related work and relevant background here. There is a long history of approaches to grammar learning. Our grammar learning technique makes use of computational grammar resources, some of

which are presented here. An essential part for our work is the concept of an abstract grammar, which we will define and explain. Furthermore, using constraint solving is a common computational approach to problem solving.

9.2.1 Previous Work on Grammar Learning

Grammar inference, i.e., the generation of a grammar from examples, has been a field of active research for quite some time. Common grammar learning approaches are:

- grammar inference, where a grammar is inferred from unannotated corpus data,
- data-oriented parsing (DOP), where subtrees from an example treebank are used as an implicit grammar,
- probabilistic context-free grammar (PCFG), where, for each rule of a context-free grammar (CFG) a probability is learned from a corpus, and
- subgrammar extraction, where a subset of syntactic rules is extracted from a more general grammar.

We draw inspiration from DOP and several of the other approaches. Most of these methods require no or very little linguistic information besides the training examples. As a result they tend to require more training data to learn more expressive grammars. The main difference of our method is, that it, in addition, requires linguistic knowledge in the form of a wide-coverage resource grammar. However, this fact allows the method to learn reasonable grammars, usually from less than 10 examples.

Grammar Inference:

There has been a lot of work on general grammar inference, both using supervised and unsupervised methods (see, e.g., overviews by (Clark and Lappin, 2010) and (D'Ulizia, Ferri, and Grifoni, 2011)). Most of these approaches have focused on CFGs, but there has also been work on learning grammars in more expressive formalisms (e.g., (Clark and Yoshinaka, 2014)).

In traditional grammar inference one starts from a corpus and learns a completely new grammar. Because the only input to the inference algorithm is an unannotated corpus, it can require a larger amount of data to learn a reasonable grammar. Clark reports results on the ATIS corpus of around 740 sentences (Clark, 2001).

DOP:

In DOP (Bod, 1992; Bod and Scha, 1997), the grammar is not explicitly inferred. Instead a treebank is seen as an implicit grammar which is used by the parser. It tries to combine subtrees from the treebank to find the most probable parse tree. The DOP model is interesting because it has some similarities with our approach, especially with the second extension where we use subtrees in

grammar learning (see Section 9.9). Bod also reports results for DOP on the Penn Treebank ATIS corpus with 750 trees (Bod and Scha, 1997).

PCFGs:

Another approach that could come to mind when thinking about learning grammars is PCFGs, an extension of context-free grammars where each grammar rule is assigned a probability. Parsing with a PCFG involves finding the most probable parse tree (Manning and Schütze, 1999, Chapter 11). The probabilities for a PCFG can be learned from annotated or unannotated corpora using the Inside-Outside algorithm, an instance of Expectation Maximazition (EM) algorithms (Lari and Young, 1990; Pereira and Schabes, 1992). The approach works for both formal and natural languages, as Pereira and Shabes Pereira and Schabes (1992) show. They present experiments involving the palindrome language $L = \{ww^R \mid w \in \{a, b\}^*\}$ and the Peen treebank and show that training on bracketed strings results in significantly better results than training on raw text. The results they report are based on 100 input sentences for the palindrome language and on the ATIS corpus using 700 bracketed sentences.

Subgrammar Extraction:

There has been some previous work on subgrammar extraction (Henschel, 1997; Kešelj and Cercone, 2007). Both articles present approaches to extract an application-specific subgrammar from a large-scale grammar focusing on more or less expressive grammar formalisms: CFG, systemic grammars (equivalent to typed unification based grammars) and HPSG. However, both approaches are substantially different from our approach, either in the input they take or in the constraints they enforce on the resulting grammar.

Logic Approaches:

To our knowledge, there have been surprisingly few attempts to use logic or constraint-based approaches, such as theorem proving or constraint optimization, for learning grammars from examples. One exception is (Imada and Nakamura, 2009), in which the authors experiment with Boolean satisfiability (SAT) constraint solvers to learn context-free grammars. They report results similar to ours, but only focus on formal languages over $\{a, b\}^*$.

9.2.2 Abstract Grammars

In our work we use the Grammatical Framework (GF). However, the way in which we state our problem allows us to substitute the grammar formalism used by any other that fulfills some basic requirements.

Grammatical Framework (Ranta, 2009b; Ranta, 2011) is a multilingual grammar formalism based on a separation between abstract and concrete syntax. The abstract level is meant to be language-independent, and every abstract syntax can have several associated concrete syntaxes, which act as language-specific realizations of the abstract rules and trees. In this chapter we only make

use of the high-level abstract syntax. This high-level view makes it possible to transfer our approach to other grammar formalisms with a comparable level of abstraction. An abstract syntax can be expressed as a many-sorted algebra (Definition 9.2.1). We build the techniques in this chapter on top of the concept of many-sorted algebras. As a consequence, the techniques work for all grammar formalisms that can be expressed as a many-sorted algebra as well.

Definition 9.2.1 (Many-sorted Algebras)**.** *A many-sorted algebra (Wirsing, 1990, pp. 680) can be defined given a signature* $\Sigma = \langle S, F \rangle$ *where* S *is a set of sorts and* F *is a set of function symbols, together with a mapping* $\textbf{\textit{type}} : F \longmapsto S^* \times S$ *which expresses the type of each function symbol.*

Notation: Instead of giving the two sets and the **type** function separately, Wirsing Wirsing (1990) gives a signature in the following form:

```
signature Σ_Bool ≡
  sort Bool
  functions
    true  : → Bool
    false : → Bool
    and   : Bool,Bool → Bool
    or    : Bool,Bool → Bool
endsignature
```

For the function types for the function symbols in a many-sorted algebra (Definition 9.2.1) the type on the right-hand side of the arrow is the result type of the function and the types on the left-hand side are the parameter types.

In this chapter we can look at grammars from various perspectives. From a linguistic point of view we talk about syntax rules, syntactic rules or grammar rules. From the perspective of algebras we talk about function. For that reason the terms rules and functions are used interchangeably. The same is the case for categories and sorts, because sorts in many-sorted algebras are used to express syntactic categories.

```
signature Σ_Example ≡
  sorts Cl, CN, VP, N, V
  functions
    PredVP    : NP, VP → Cl
    DetCN     : Det, CN → NP
    UseN      : N → CN
    man_N     : → N
    sleep_V   : → V
    detSg_Det : → Det
endsignature
```

Figure 9.1: Signature of a many-sorted algebra for an example abstract grammar

The many-sorted algebra defined by the signature in Figure 9.1 can be expressed in GF syntax as shown in Figure 9.2. We do not want to go into detail about the GF abstract syntax (see e.g. (Ranta, 2011) for more details).

```
abstract Example = {
  flags startcat = Cl ;
  cat Cl; VP; Det; N; V;
  fun PredVP : NP -> VP -> Cl ;
      DetCN : Det -> CN -> NP ;
      UseN : N -> CN ;
      UseV : V -> VP ;
      man_N : N ;
      sleep_V : V ;
      theSg_Det : Det ;
}
```

Figure 9.2: Example abstract syntax in Grammatical Framework for the abstract grammar in Figure 9.1

Instead we will express all relevant grammars in this chapters as signatures of many-sorted algebras.

To make the grammar in Figure 9.1 more accessible, we give a short explanation of the rules. This grammar has three constant functions: `man_N` representing the noun *man*, `sleep_V` representing the intransitive verb *sleep* and `detSg_Det` representing the definite article in singular *the*. The function `UseN` converts a noun into a common noun, i.e., a noun phrase without determiner, and `DetCN` adds the determiner to form a complete noun phrase. The function `UseV` forms a verb phrase from an intransitive verb. Finally, `PredVP` combines a subject noun phrase and a verb phrase into a declarative clause.

Base on such abstract grammars we can define abstract syntax trees (Definition 9.2.2). There is no formal difference between leaves and internal nodes, but a leaf is just a node without any children. The trees are therefore similar to abstract syntax trees as they are used in, e.g., computer science (Ranta, 2012, Chapter 2.5).

Definition 9.2.2 (Abstract Syntax Tree)**.** *Given a signature of a many-sorted algebra $\Sigma = \langle S, F \rangle$, an abstract syntax tree is either*

- *A node $n : C$ without children and with $n \in F$, $C \in S$ and $\boldsymbol{type}(n) = C$*
- *A node $n : C$ with $n \in F$, $T \in S$ and $\boldsymbol{type}(n) = (C_1, \ldots, C_i, T)$, and the node has i valid abstract syntax trees with root nodes $c_1 : C_1, \ldots, c_i : C_i$ as children*

As an example, an abstract representation of the English sentence *The man sleeps* can be seen in Figure 9.3. The tree is valid according to the grammar $\Sigma_{Example}$ (Figure 9.1) because it follows Definition 9.2.2.

9.2.3 Wide-Coverage and Resource Grammars

For various grammar formalisms there exist large grammars describing significant parts of natural languages. Examples include the HPSG resource grammars developed within the DELPH-IN collaboration (DELPH-IN, 2020) or grammars created for the XMG metagrammar compiler (XMG, 2017) .

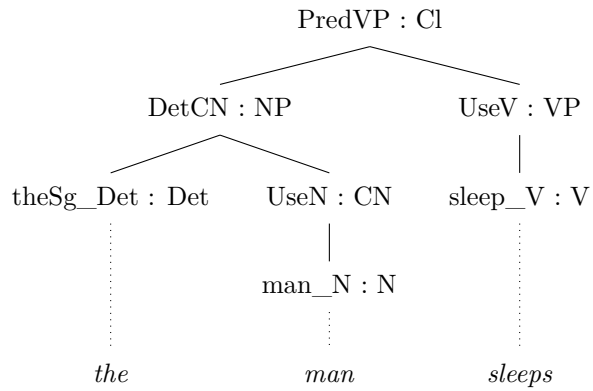


Figure 9.3: Abstract syntax tree for the example sentence *the man sleeps* – note that the English surface words and the dotted edges are not part of the abstract syntax

The resource grammar available in GF is called the GF resource grammar library (RGL) (Ranta, 2009b), which is a general-purpose grammar library for more than 30 languages which covers a wide range of common grammatical constructions. The main purpose of the RGL is to act as a application programming interface (API) when building domain-specific grammars. It provides high-level access to the linguistic constructions, facilitating development of specific applications. The inherent multilinguality also makes it easy to create and maintain multilingual applications.

However, it is necessary to learn the GF formalism as well as to write grammars in general to use the RGL for developing GF grammars. This limits the user group drastically. In contrast, the methods presented in this chapter allow non-grammarians to create grammars for their own domain or application without the knowledge of the grammar formalism and methods of grammar writing.

9.2.4 Constraint Satisfaction Problems

Many interesting problems can be formulated as constraint satisfaction problems (CSP) (Russell and Norvig, 2009, chapter 6). For a CSP, one targets to find an assignment of a number of variables that respect some given constraints. CSPs are classified depending on the domains of the variables, and the kinds of constraints that are allowed.

If an objective function is added to a CSP to add a judgment of optimality, we talk of a constraint optimization problem (COP) instead. In this chapter we formulate our problem based on Boolean variables in the constraints, but require integer operations in the objective functions. An objective function is the function whose value has to be maximized or minimized while solving the constraints.

We use the IBM ILOG CPLEX Optimization Studio[2] to find solutions to this integer linear programming (ILP) restricted to 0/1 integers. Other solvers such as the free and open source solver GLPK (GNU Linear Programming Kit)[3] can be used as well, but, currently the free alternatives suffer from larger performance issues.

9.3 Learning a Subgrammar

In this section we will describe the task and the problem to be solved. We start from a large, expressive, but over-generating, resource grammar and a small set of example sentences. From this input we want to infer a subgrammar that covers the examples and is optimal with respect to some objective function. One possible objective function would be, e.g., to reduce the number of syntactic analyses per sentence.

9.3.1 Subgrammar Extraction by Tree Selection

We assume that we already have a parser for the resource grammar that returns all possible parse trees for the example sentence. That means we can start from a set of trees for each sentence. From the syntax trees we can extract the list of syntactic functions involved (Definition 9.3.1). The grammar we want to learn still has to be able to cover all these sentences and should be optimal according to some optimality criterion. We formulate our problem in Definition 9.3.3.

Definition 9.3.1 (Flattened Abstract Syntax Tree)**.** *Given an abstract syntax tree T, we can define a flattened representation as the set*

$$T_{flat} = \{n \mid n : T \text{ is a node in the abstract syntax tree}\}$$

The resulting representation loses all structural and type information but is sufficient for our purposes (see Lemma 1 in Section 9.3.2).

Definition 9.3.2 (Subgrammar)**.** *Given a many-sorted algebra with signature $\Sigma = \langle S, F \rangle$, a subgrammar is a many-sorted algebra given by the signature $\Sigma' = \langle S', F' \rangle$ with $S' \subseteq S$, $F' \subseteq F$ and for all $f \in F'$: $\boldsymbol{type}_{\Sigma'} = \boldsymbol{type}_{\Sigma} = (C_1, \ldots, C_n)$*

Definition 9.3.3 (Subgrammar Learning Problem)**.**

- *Given: $\mathcal{F} = \{F_1, \ldots, F_n\}$, a set of forests where each forest $F_k = \{T_{k1}, ..., T_{kt_k}\}$, t_k is the number of trees in F_k, and each forest F_k is a syntactic representation of the example sentence s_k.*
- *Problem: select at least one T_{ki_k} from each F_k, while minimizing the objective function.*

[2] `http://www.cplex.com/`

[3] `https://www.gnu.org/software/glpk/`

Possible objective functions for our problem include:

rules the number of rules in the resulting grammar (i.e., reducing the grammar size);

trees the number of all initial parse trees T_{ki} that are, intended or not, valid in the resulting grammar (i.e., reducing the syntactic ambiguity);

rules+trees the sum of **rules** and **trees**;

weighted a modification of **rules+trees** where each rule is weighted by the number of occurrences in all F_k.

The problem we describe here is an instance of a set covering problem. A related problem is the Hitting Set problem (Garey and Johnson, 1979, section A3.1), which is NP-complete (Karp, 1972). Since our problem is a generalization of the Hitting Set problem, it is NP-complete as well.

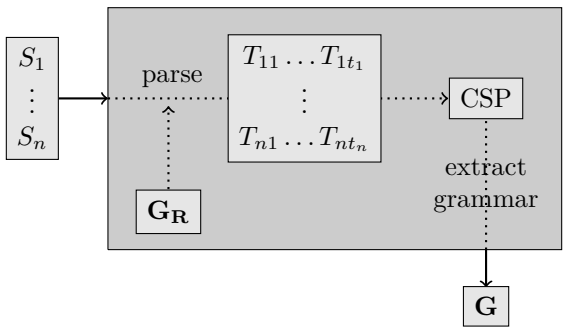


Figure 9.4: The outline of our grammar learning system

9.3.2 Modeling Subgrammar Extraction as a Constraint Problem

Even though there exist other solutions to the related class of set covering problems, a natural approach to this problem is to model it as a constraint satisfaction problem.

An outline of the system architecture is shown in Figure 9.4. Given the set of trees for each sentence, there are various possible ways to model the problem, depending on the choice of the atomic units we want to represent by the logic variables. The logic variables can encode subtrees of various sizes, ranging from subtrees of size 1, i.e., single nodes or syntactic functions, to subtrees of arbitrarily larger sizes. There are also different ways to split a tree into these larger subtrees. In the following, we use subtrees of size 1, but see Section 9.9 for an extension to larger subtrees.

As a result, we can represent an abstract syntax tree T_{ki} as the set of labels in the tree, $T_{\text{flat}_{k_i}} = \{r_1, \ldots r_n\}$ (Definition 9.3.1). This results in a loss of structural information but does not have any negative effect on the outcome of

our approach. Another possible representation would be multi-sets, but it can be easily shown that this would not result in any improvement (Lemma 1).

An example can be seen in Figures 9.5–9.6. We start with Figure 9.5 and go from left to right, starting with the sentences. Each sentence results in one or several syntax trees, which than can be represented in a flattened form (Definition 9.3.1).

Lemma 1. *For an abstract syntax tree T, when encoding it as a conjunction of variables representing rules, the result is the same when flattening according to Definition 9.3.1 using a set or when encoding it as a multi-set instead.*

Proof.

- In a disjunction of variables, variables can be reordered and repeating variables can be eliminated, i.e., $x_1 \vee x_2 \equiv x_2 \vee x_1$ and $x \vee x \equiv x$.
- When translating a multi-set into a disjunction, variables for rules that occur several times can be eliminated.
- The resulting formula is equivalent to the translation of the set representation.

□

The resulting constrains can be seen in Figure 9.6. We have variables for each sentence, each tree and all the syntax rules occurring in the trees. First we enforce that all sentences have to be covered, then we describe for each sentence what trees have to be covered and finally for each tree, if it should be covered, what rules have to be covered.

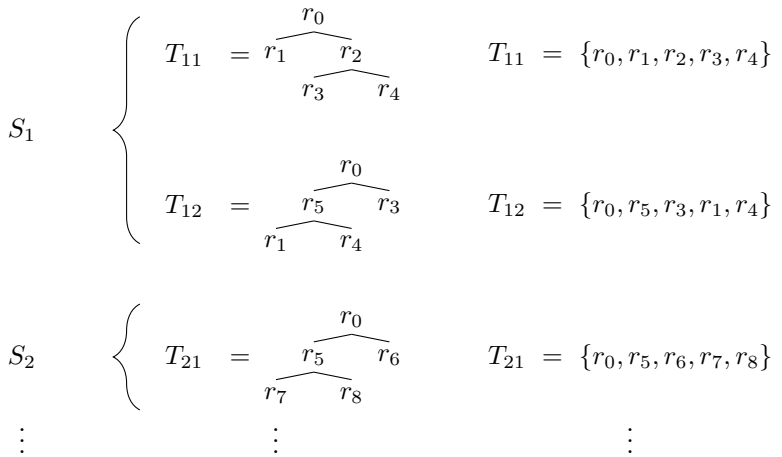
The solution to this problem gives rise to a new grammar, which, following the definition of our subgrammar learning problem (Definition 9.3.3), also covers all examples. Lemma 2 shows that the resulting grammar is again a many-sorted algebra and according to Lemma 3 it is also valid subgrammar of the original grammar following Definition 9.3.2.

Lemma 2. *From the solution to the CSP and the original grammar $\mathbf{G_R}$ (defined by $\Sigma_{\mathbf{G_R}}$), a new grammar $\mathbf{G}$, i.e., a new many-sorted algebra $\Sigma_{\mathbf{G}}$, can be constructed*

Proof. The CSP solution is the union of the sets of variables with value 1 for sentences (S), trees (T) and rules (R). We are only interested in the set R representing the rules included in the solution.
The new grammar has the signature $\Sigma_{\mathbf{G}} = \langle S_{\mathbf{G}}, F_{\mathbf{G}} \rangle$, where:
(1) The set of sorts can be defined as

$$S_{\mathbf{G}} = \bigcup_{r \in R} \{C_1, \ldots, C_n \mid \mathbf{type}_{\mathbf{G_R}}(r) = (C_1, \ldots, C_n)\}$$

(2) The set of functions can be simply defined as $F_{\mathbf{G}} = R$.
(3) The typing function is defined as $\mathbf{type}_{\mathbf{G}} = \mathbf{type}_{\mathbf{G_R}} \restriction_{F_{\mathbf{G}}}$, i.e. the restriction of the original typing function $\mathbf{type}_{\mathbf{G_R}}$ to the set $F_{\mathbf{G}}$. □



(a) Sentences (b) Trees (c) Flattened representations

Figure 9.5: Sentences and tree representations

All sentences have to be covered:

$S_1 \wedge S_2 \wedge \ldots$

At least one tree per sentence has to be covered:

$S_1 \rightarrow T_{11} \vee T_{12}$

$S_2 \rightarrow T_{21}$

$\ldots$

All rules in a tree have to be covered:

$T_{11} \rightarrow r_0 \wedge r_1 \wedge r_2 \wedge r_3 \wedge r_4$

$T_{12} \rightarrow r_0 \wedge r_5 \wedge r_3 \wedge r_1 \wedge r_4$

$T_{21} \rightarrow r_0 \wedge r_5 \wedge r_6 \wedge r_7 \wedge r_8$

$\ldots$

Figure 9.6: Encoding of Figure 9.5 as logical constraints

Lemma 3. *The new grammar* $\mathbf{G}$ *described by the signature* $\Sigma_{\mathbf{G}}$ *is a subgrammar of the original grammar* $\mathbf{G_R}$ *given by the signature* $\Sigma_{\mathbf{G_R}}$

Proof. By definition, $\mathbf{G}$ is a subgrammar of $\mathbf{G_R}$ if the following holds:

$F_{\mathbf{G}} \subseteq F_{\mathbf{G_R}}$ By definition of $F_{\mathbf{G}}$ in Lemma 2 (2) and the definition of the learning problem (Definition 9.3.3) and abstract syntax trees (Definition 9.2.2)

$S_{\mathbf{G}} \subseteq S_{\mathbf{G_R}}$ By definition of $S_{\mathbf{G}}$ in Lemma 2 (1) and the definition of the typing function on $\mathbf{G_R}$, $\mathbf{type}_{\mathbf{G_R}} : F \longmapsto S_{\mathbf{G}} * \times S_{\mathbf{G}}$ (Definition 9.2.1)

$\forall f \in F_{\mathbf{G}} : \mathbf{type}_{\mathbf{G}}(f) = \mathbf{type}_{\mathbf{G_R}}(f)$ By definition of $\mathbf{type}_{\mathbf{G}}(f)$ and function restriction

□

```
laula laulu
sing a song
PhrUtt NoPConj (UttImpSg PPos (ImpVP (ComplSlash (SlashV2a sing_V2)
  (DetCN (DetQuant IndefArt NumSg) (UseN song_N))))) NoVoc

laulakaa laulu
sing a song
PhrUtt NoPConj (UttImpPl PPos (ImpVP (ComplSlash (SlashV2a sing_V2)
  (DetCN (DetQuant IndefArt NumSg) (UseN song_N))))) NoVoc
...
minä haluan laulaa laulun suihkussa
I want to sing a song in the shower
PhrUtt NoPConj (UttS (UseCl (TTAnt TPres ASimul) PPos (PredVP
  (UsePron i_Pron) (ComplVV want_2_VV (AdvVP (ComplSlash
  (SlashV2a sing_V2) (DetCN (DetQuant IndefArt NumSg) (UseN song_N)))
  (PrepNP in_Prep (DetCN (DetQuant DefArt NumSg) (UseN shower_N))))))))) NoVoc
```

Figure 9.7: Excerpt from the Finnish treebank used in the "Comparing-Against-Treebank" experiment. The Finnish example is followed by the English translation and the abstract syntax tree. Sources of (morpho)syntactic ambiguity are highlighted.

9.4 Bilingual Grammar Learning

If our example sentences are translated into another language, and the resource grammar happens to be bi- or multilingual[4], we can use that knowledge to improve the learning results.

This can be relevant because various languages express different features explicitly. As an example, consider Figure 9.7: Finnish does not express definite or indefinite articles, so *laula laulu* can be translated to both *sing a song* and *sing the song*. On the other hand, the verb *sing* in the English imperative phrase *sing a song* is morphosyntactically ambiguous – it can be singular or plural,

[4]The same abstract grammar is used to describe multiple languages in parallel

while Finnish makes the distinction into *laula laulu* (singular) and *laulakaa laulu* (plural). This example shows how English can be used to disambiguate Finnish and vice versa.

For each sentence pair (S_i, S'_i), we parse each sentence separately using the resource grammar into the forests $F_i = \{T_{i1} \dots T_{it_i}\}$ and $F'_i = \{T'_{i1} \dots T'_{it'_i}\}$. We then only keep the trees that occur in both forests, i.e., $F_i \cap F'_i$. These filtered tree sets are translated into logical formulas, just as for monolingual learning.

The intersection of the trees selects the (morpho)syntactically disambiguated reading. The disambiguation makes the constraint problem smaller and the extracted grammar more likely to be the intended one.

9.5 Implementation

We have implemented the system we outlined in Figure 9.4 and the previous section as well as all aspects of the following evaluation and extensions. The implementation is done in Haskell and released as open source.[5] As constraint solvers both GLPK and, if available, CPLEX can be used.

The system can be treated as a black box that takes a set of sentences $S_1 \dots S_n$ as an input and produces a grammar $\mathbf{G}$ as output, doing so by relying on a resource grammar (labeled $\mathbf{G_R}$).

First the sentences are parsed using the resource grammar $\mathbf{G_R}$ and the syntax trees translated into logical formulas, in the way described in Section 9.3.2. The logical formulas are then translated into ILP constraints and handed over to the constraint solver, which returns a list of rule labels that form the basis for the new restricted grammar $\mathbf{G}$. The output of the solver is influenced by the choice of the objective function (candidates are described in Section 9.3.1).

The translation between logical formulas and linear inequalities is well established and an example can be seen in Figure 9.8. Conjunctions and disjunctions are converted into sums. The direction of the inequality as well as the multiplication constant are chosen accordingly, depending on if it is an implication, a conjunction or a disjunction.

In fact, the solver does not necessarily only return one solution. In case of several solutions, they are ordered by the objective value. Choosing the one with the best value is a safe choice even though there might be a solution with a slightly worse score that actually performs better on the intended task.

[5] `https://github.com/MUSTE-Project/subgrammar-extraction`

All sentences have to be covered:

$$S_1 \wedge S_2 \wedge \cdots \wedge S_n$$

Each sentence has at least one tree:

$$S_1 \rightarrow T_{11} \vee T_{12}$$
$$S_2 \rightarrow T_{21}$$
$$\ldots$$

All rules in a tree have to be covered:

$$T_{11} \rightarrow r_0 \wedge r_1 \wedge r_2 \wedge r_3 \wedge r_4$$
$$T_{12} \rightarrow r_0 \wedge r_5 \wedge r_3 \wedge r_1 \wedge r_4$$
$$T_{21} \rightarrow r_0 \wedge r_5 \wedge r_6 \wedge r_7 \wedge r_8$$
$$\ldots$$

(a) Logical formulas

All sentences have to be covered:

$$n \leq S_1 + S_2 + \ldots S_n$$

Each sentence has at least one tree:

$$S_1 \leq T_{11} + T_{12}$$
$$S_2 \leq T_{21}$$
$$\ldots$$

All rules in a tree have to be covered:

$$5 \times T_{11} \leq r_0 + r_1 + r_2 + r_3 + r_4$$
$$5 \times T_{12} \leq r_0 + r_5 + r_3 + r_1 + r_4$$
$$5 \times T_{21} \leq r_0 + r_5 + r_6 + r_7 + r_8$$
$$\ldots$$

(b) ILP constraints

Figure 9.8: Translation between logical formulas and ILP constraints

9.6 Evaluation

Related literature (D'Ulizia, Ferri, and Grifoni, 2011) suggests several measures for the performance of grammar inference algorithms, most prominently the methods "Looks-Good-To-Me", "Rebuilding-Known-Grammar" and "Compare-Against-Treebank". Our inferred grammars passed the informal "Looks-Good-To-Me" test, so we designed two experiments to demonstrate the learning capabilities of our approach following the other two methods.

9.6.1 Rebuilding a Known Grammar

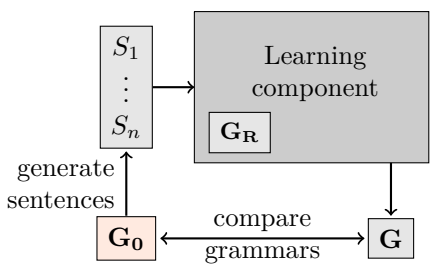


Figure 9.9: Setup for an evaluation by rebuilding a known grammar $\mathbf{G_0}$

The evaluation process is shown in Figure 9.9. It is based on the learning component (Figure 9.4), which is highlighted. To evaluate our technique in a quantitative way we start with two grammars $\mathbf{G_R}$ and $\mathbf{G_0}$, where $\mathbf{G_0}$ is a subgrammar of $\mathbf{G_R}$. We use $\mathbf{G_0}$ to generate random example sentences S_1, ...,

S_n. These examples are then used to learn a new grammar $\mathbf{G}$ as described in Section 9.5. The aim of the experiment is to see how similar the inferred grammar $\mathbf{G}$ is to the original grammar $\mathbf{G_0}$. To measure this similarity, we compute precision and recall in the following way, where $F_{\mathbf{G_0}}$ are the rules of the original grammar and $F_{\mathbf{G}}$ the rules of the inferred grammar:

$$Precision = \frac{|F_{\mathbf{G_0}} \cap F_{\mathbf{F}}|}{|F_{\mathbf{G}}|} \qquad Recall = \frac{|F_{\mathbf{G_0}} \cap F_{\mathbf{G}}|}{|F_{\mathbf{G_0}}|}$$

We can analyze the learning process depending on, e.g., the number of examples, the size of the examples and the language involved.

We conducted this experiment for Finnish, German, Swedish, Spanish and English. For each of these languages we used the whole GF RGL as the resource grammar $\mathbf{G_R}$ and a small subset containing 24 syntactic and 47 lexical rules as our known grammar $\mathbf{G_0}$ (Figure 9.10).

```
signature Σ_Example ≡
  sorts
    A, Adv, Ant, AP, Cl, Comp, CN, Conj, CN, Det, ListNP,
    N, N2, Num, PN, Prep, Pron, Quant, S, Temp, Tense,
    Utt, V, V2, VA, VP, VPSlash
  functions
    UseN : N → CN
    UseN2 : N2 → CN
    AdjCN : AP, CN → CN
    UsePN : PN → NP
    UsePron : Pron → NP
    DetCN : Det, CN → NP
    AdvNP : NP, Adv → NP
    ConjNP : Conj, ListNP → NP
    BaseNP : NP, NP → ListNP
    PositA : A → AP
    PrepNP : Prep, NP → Adv
    UseV : V → VP
    ComplSlash : VPSlash, NP → VP
    SlashV2a : V2 → VPSlash
    ComplVA : VA, AP → VP
    AdvVP : VP, Adv → VP
    PredVP : NP, VP → Cl
    UseCl : Temp, Pol, Cl → S
    UttS : S → Utt
    AdvS : Adv, S → S
    UseComp : Comp → VP
    CompAP : AP → Comp
    TTAnt : Tense, Ant → Temp
    DetQuant : Quant, Num → Det
endsignature
```

Figure 9.10: Signature of many-sorted algebra used as $\mathbf{G_0}$

We tested the process with an increasing number of random example sentences (from 1 to 20), an increasing maximum depth of the generated syntax trees (from 6 to 10), five languages (Finnish, German, Spanish, Swedish and English), and our four different objective functions from Section 9.3.1.

9.6.2 Comparing Against a Treebank

Our second approach to evaluate our grammar learning technique has a more manual and qualitative focus, and is depicted in Figure 9.11 (again with the highlighted learning component). Instead of starting from a grammar which we want to rebuild, we start from a treebank (S_1, T_1), ..., (S_n, T_n), i.e., a set of example sentences in a language and one gold-standard tree for each sentence.

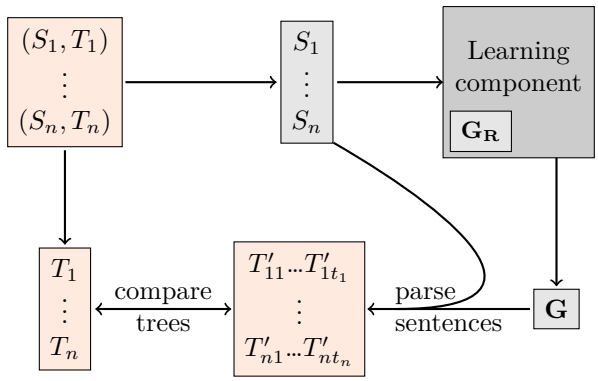


Figure 9.11: Setup for an evaluation by comparing the learned grammar to a treebank

We use the plain sentences S_1, ..., S_n from the treebank to learn a new grammar $\mathbf{G}$, using the GF RGL, extended with the required lexicon, as the resource grammar $\mathbf{G_R}$. Then the system parses the sentences with the resulting grammar $\mathbf{G}$, and compares the resulting trees with the original trees in our gold standard. If the original tree T_i for sentence S_i is among the parsed trees T'_{i1}, ..., T'_{in}, it is reported as a success.

If the gold standard tree is not covered, we could use a more fine-grained similarity score, such as labeled attachment score (LAS) or tree edit distance. However, because of the limited size of the treebanks we decided against this evaluation measure.

The data we used for testing the grammar learning consists of hand-crafted treebanks for the following languages: Finnish, German, Swedish and Spanish (see Table 9.1 for statistics and Listing 9.7 for a fragment of the Finnish treebank). We exclude English here because we use it as the second language in bilingual learning on the treebank in the next section.

9.6.3 Comparing Against a Bilingual Treebank

Our final experiment is a repetition of the "Compare-Against-Treebank" experiment, but using a bilingual treebank instead of a monolingual one.

The treebanks we created contain English translations of all sentences. This means we have access to four bilingual treebanks Finnish-English, Spanish-English, Swedish-English and German-English. We used the bilingual learning component described in Section 9.4, using the GF RGL which is a multi-lingual resource grammar covering all these languages.

9.7 Results

We conducted the experiments described in the previous section and got very promising results. In this section we will discuss the results in detail.

9.7.1 Results: Rebuilding a Known Grammar

We ran the first experiment, described in Section 9.6.1, and a selection of the results can be seen in Figures 9.12–9.14. Out of the many possible experiments (5 languages, 4 objective functions and 5 different tree depths for generating examples) we present 3 representative samples:

- the same objective function and tree depth with various languages,
- the same language and tree depth with various objective functions, and
- the same language and objective function with various tree depths.

We report precision and recall for a sequence of experiments, where for each experiment we generated sets of random sentences with increasing size.

All three graphs (Figure 9.12, 9.13 and 9.14) resemble typical learning curves where the precision stays mostly stable while the recall rises strongly in the beginning and afterwards approaches a more or less stable level. The precision rises slightly between 1 and 5 input sentences. The recall remains almost constant after input of about 5 sentences. With larger input the precision starts to drop slightly when the system learns additional rules that are not part of the original grammar. These curves are pretty much stable across all languages (see Figure 9.12), objective functions (see Figure 9.13) and maximum tree depth used in sentence generation (see Figure 9.14), and show that we get the best results with about 10 examples.

One exception can be seen in Figure 9.14. With a maximum tree depth of 5 the system can only achieve a recall of about 0.7, which means that for this tree depth it does not encounter all grammar rules.

These results confirm that our method is very general and provides good results, especially for really small training sets of only a few to a few dozen sentences. By starting from a linguistically sound source grammar, which our learning technique recovers by extracting a subgrammar, we can show that the learned grammar is sound in a similar way.

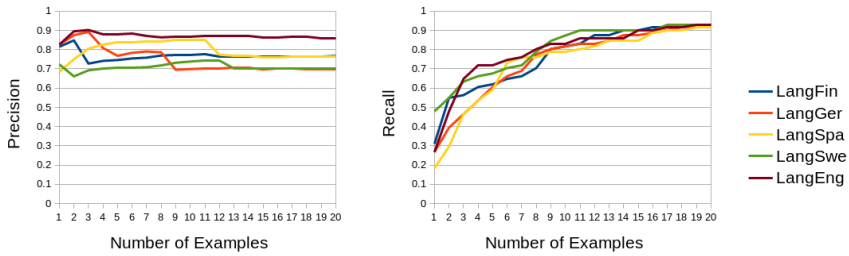


Figure 9.12: Results for objective function **rules**, maximum tree depth 9 and various languages

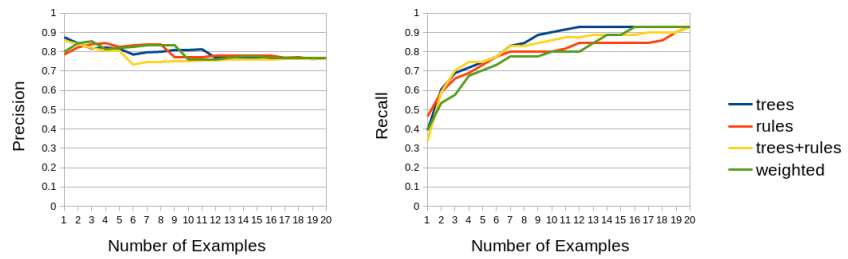


Figure 9.13: Results for Finnish, maximum tree depth 9 and various objective functions

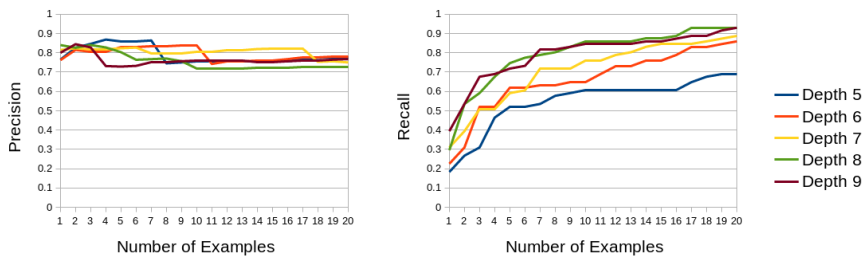


Figure 9.14: Results for English with objective function **rules** and various generation depths

9.7.2 Results: Comparing Against a Treebank

We used the treebanks and the process described in Section 9.6.2 to further evaluate our learning method. Table 9.1 shows the results of running our experiment on monolingual and bilingual treebanks of four different languages, and with two objective functions, **rules+trees** and **weighted**. The table columns are:

Size the number of sentences in the treebank

Accuracy the accuracy, meaning the percentage of sentences where the correct tree is among the parse trees for the new grammar

Ambig. the syntactic ambiguity, i.e., the average number of parse trees per sentence

The system can cover all the sentences of the treebank with our learned grammar and, as the table shows, in most cases the results include the gold standard tree. We inspected more closely the sentences where the grammar fails to find the gold standard tree, and found that the trees usually differ only slightly, so if we used attachment scores instead we would get close to 100% accuracy in every case.

A clear exception is the case of the monolingual Finnish treebank. When we use the **rules+trees** objective function, we have serious problems learning the correct grammar, with only 1 correct sentence out of 22. This is due to a high level of morphosyntactic ambiguity among Finnish word forms. If we instead use the **weighted** objective function, we get a decent accuracy, but the grammar becomes highly syntactic ambiguous with 115 parse trees per sentence on average. The second part of the experiment, using a bilingual treebank, solves most of the problems involving Finnish while also improving results for other languages.

9.7.3 Results: Using Bilingual Treebanks

When we repeated the previous experiment using translation pairs as described in Section 9.6.3 we got very similar results for most of the languages, as can be seen on the right side of Table 9.1. The main difference is that the resulting grammars are more compact for the **weighted** objective function, resulting in fewer analyses. Notably, for Finnish the average number of trees per sentence drops by one order of magnitude. This is because the high level of syntactic ambiguity of Finnish sentences is reduced when disambiguated using the English translations.

		rules+trees		**weighted**	
	Size	Accuracy	Ambig.	Accuracy	Ambig.
Finnish	22	5%	1.0	91%	115
German	16	75%	1.1	100%	2.0
Swedish	10	100%	1.1	100%	2.8
Spanish	13	100%	1.2	92%	3.7

(a) Monolingual treebank

		rules+trees		**weighted**	
	Size	Accuracy	Ambig.	Accuracy	Ambig.
Finnish	22	86%	4.9	96%	8.7
German	16	94%	1.1	100%	1.5
Swedish	10	100%	1.1	100%	1.2
Spanish	13	100%	1.2	100%	2.3

(b) Bilinguallingual treebank

Table 9.1: Results for comparing against a treebank. Accuracy means the percentage of sentences where the correct tree is found, and Ambig(uity) means the average number of parse trees per sentence.

9.8 Extension 1: Negative Examples

The first addition to the original grammar learning method, to which we dedicated the previous sections, is to add negative examples to the learning process. Negative examples can speed up the learning process for certain syntactic constructions by narrowing down the grammar, e.g., by using positive and negative examples that are minimal pairs concerning the intended linguistic phenomenon.

To allow for negative examples, i.e., example sentences that should not be parsable in the new grammar, we have to add additional constraints. These new constraints have to express that, for each of the syntax trees we get from the negative example sentences, at least one rule, of those involved in the parse of the negative example, has to be excluded in the learned grammar. Or conversely, that not all rules can be included. As a logic formula, this results in the negation of the conjunction of all rules for a tree, i.e.,

$$\neg r_1 \vee \neg r_2 \vee \cdots \vee \neg r_n \;\equiv\; \neg(r_1 \wedge r_2 \wedge \cdots \wedge r_n)$$

This simple addition allows negative examples in the basic constraint optimization problem described in Section 9.3.2. We will demonstrate in two examples how this feature can be used.

9.8.1 Examples

We can demonstrate how positive and negative examples can be used together, in general, by having a look at both formal languages and fragments of natural languages.[6]

9.8.1.1 Dyck Language

The Dyck language is a language of balanced opening and closing parentheses,

$$\mathcal{L}_{\text{Dyck}} = \{w \in V^* \mid \text{Each prefix of w contains no more)'s than ('s and there are exactly as many ('s as)'s in w}\}$$

with $V = \{(,)\}$[7] In our example we extend this to two kinds of bracketing symbols, parenthesis "( )", and brackets "[ ]", i.e., $V_{\text{Dyck}} = \{(,),[,]\}$. The language $\mathcal{L}_{\text{Bracket}} = \{w \mid w \in V^*_{\text{Dyck}}\}$, the language of all strings over the alphabet V_{Dyck}, can be expressed with the grammar in Figure 9.15. The semantics of the rules is the following:

`Empty` introduces an empty string

`LeftP` (or `RightP`) introduces a single left (or right) parenthesis

`LeftS` (or `RightS`) introduces a single left (or right) square bracket

`BothP` (or `BothS`) wraps a balanced pair of parentheses (or square brackets) around a string

`Conc` concatenates a pair of strings

Learning the Dyck language of balanced parentheses from the grammar in Figure 9.15, using examples, can be either quite trivial or pretty tricky. With the objective function minimizing the number of rules, the grammar learning technique immediately outputs the intended grammar from just the positive examples "( )", "[ ]" and "( ) ( )". However, with the objective function minimizing the number of parse trees the technique learns a wrong grammar, allowing unbalanced parentheses.

By adding the negative examples "( ]" and "[ )" as well as "(" and "[" we solve the issue and the learning component provides the correct grammar. The resulting grammar in Figure 9.16 is a subset of the original grammar in Figure 9.15.

This example demonstrates how sometimes adding negative examples can help us learn a grammar in a quick and immediate way.

[6] The examples are in English but the problems we approach are language independent.

[7] Usually the alphabet is denoted with the letter Σ. to avoid naming conflicts with signatures, we use the letter V instead.

9.8.1.2 Adverbials

In the previous section we showed how negative examples can help to learn artificial formal languages. The next step is to show that the same applies to natural languages.

To show how quickly the proposed technique of grammar learning can learn a desired subgrammar, we have to start from a wide-coverage grammar. We use the full GF RGL again.

One of the problems that can be solved with negative examples is the handling of adverbials in the RGL. A relevant fragment of the RGL is shown in Figure 9.17. Various syntactic constructions such as prepositional phrases, created by the function `PrepNP`, are assigned the syntactic category `Adv`. Furthermore, almost every part of speech can be modified by adverbials, such as noun phrases using `AdvNP` and verb phrases using `AdvVP`

This can lead to syntactic ambiguity when an adverb or adverbial, potentially modifying two different parts of speech, appears in the same position of the sentence. Despite human intuition, the sentence *the boy reads a book today* is, according to the syntactic functions in the RGL, syntactically ambiguous and has two different readings (see Figure 9.18).

One solution to the problem would be to add more positive examples to learn only one of the alternatives. Another solution is to add a negative example that simply rules out one of the readings.

In this case such a negative example could be **a book today arrives*. It only has the undesired reading of attaching the adverb to the noun phrase. Together with the positive example it can be used to disambiguate the readings and only the intended first reading where the adverb modifies the verb phrase remains.

In a similar way simple cases of syntactic ambiguity can be resolved using negative examples. For more advanced cases, e.g., to distinguish between lexical adverbs as adverbials and prepositional phrases in the same context, we need other mechanisms (see, e.g., Section 9.9).

9.8.2 Iterative Grammar Learning Process

The inclusion of negative examples allows us to create a human-centric and example-based grammar learning and refinement system.

Starting from a wide-coverage grammar, the user can give a set of example sentences. From these examples a first version of the domain-specific grammar can be learned. This domain-specific grammar will then be extended and refined iteratively.

The user can either give more examples to extend the grammar or ask the system for example sentences. These examples can either be marked as acceptable or as erroneous. When a sentence is marked as wrong, it is added as a negative example and the learning process starts again with the new examples. This way negative examples can be used to step-wise refine the grammar.

Because this process is purely example-based, no knowledge about grammar engineering is required. This means that a wide range of people can use the system, not only linguists. This is especially relevant in use-cases where the

```
signature Σ_Dyck
  sort Dyck
  functions
    Empty : Dyck
    LeftP, RightP, LeftS, RightS : → Dyck
    BothP, BothS : Dyck → Dyck
    Conc : Dyck, Dyck → Dyck
endsignature
```

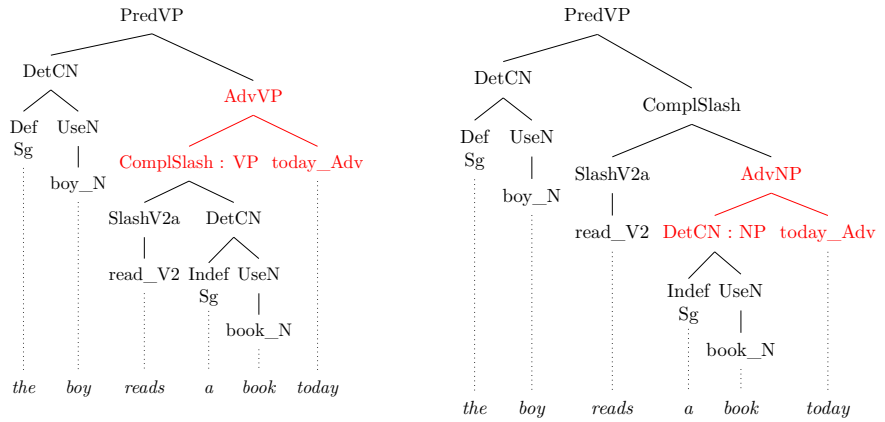
Figure 9.15: Signature of the over-generating Dyck language

```
signature Σ_Dyck' ≡
  sort Dyck
  functions
    Empty : → Dyck
    BothP, BothS : Dyck → Dyck
    Conc  : Dyck, Dyck → Dyck
endsignature
```

Figure 9.16: Signature of the resulting grammar to cover the Dyck language

```
signature Σ_RGL ≡
  sorts
    Adv, NP, Prep, VP, …
  functions
    AdvNP  : NP, Adv → NP
    AdvVP  : VP, Adv → VP
    PrepNP : Prep, NP → Adv
    …
endsignature
```

Figure 9.17: Signature of a fragment of the GF RGL concerning adverbials



(a) The adverb attached to the verb phrase

(b) The adverb attached to the object noun phrase

Figure 9.18: The two syntactic analyses according to the RGL for *the boy reads a book today*

people who are involved in creating grammars are specialists in other fields, such as teachers in a language learning setup or healthcare professionals building communication support in their field.

9.9 Extension 2: Extracting Subtrees as Basic Units

The second modification is the generalization from syntactic rules as the atomic units to subtrees. Together with a method to merge syntactic rules into new, more specific rules, this allows to address some shortcomings of the technique presented in the previous sections. The idea behind this extension is similar to the approach used by Bod Bod (1992) for data-oriented parsing (DOP). In DOP, the use of larger subtrees resulted in significantly better learning results (Bod and Scha, 1997).

To be able to use subtrees as basic units, we need to be able to create a constraint satisfaction problem of a similar structure as we used before to formulate our original constraint optimization problem, i.e., similar to a many-sorted algebra (see Section 9.2.2). Instead of converting the syntax trees into lists of syntax rules to be converted into logical variables, we can split the syntax trees into all possible subtrees up to a certain size. The splitting happens in a way that we get a list of splits and each split contains only subtrees that can be reassembled to the original tree. This is necessary to guarantee that the inferred grammar can still cover all the example sentences (Figure 9.19).

Because our grammars are equivalent to many-sorted algebras and our syntax rules are similar to functions in mathematics, we can combine several of them to a new function using function composition. For example, if we have

the two rules `PredVP : NP, VP → Cl` and `UseV : V → VP`. we can combine them into a new rule `PredVP#?#UseV : NP, V → Cl`. The resulting structure is again a many-sorted algebra.

The main motivation for using subtrees as the basic units is that when merging subtrees into new grammar rules, we can create more precise and specific grammars than the wide-coverage grammar. This also means that we step away from pure subgrammar extraction into creating more independent grammars.

Having the splits into subtrees, we can translate the splits into logical variables. The procedure follows along the same lines as it worked for syntax rules. However, one additional level has to be introduced. Previously, to cover a tree, all its rules had to be covered. Now we have the additional level of splits. That means, to cover a tree, at least one of the splits has to be covered, and to cover a split, all its subtrees have to be covered. So for the example in Figure 9.19, to cover the tree, we end up with the following constraint involving the splits:

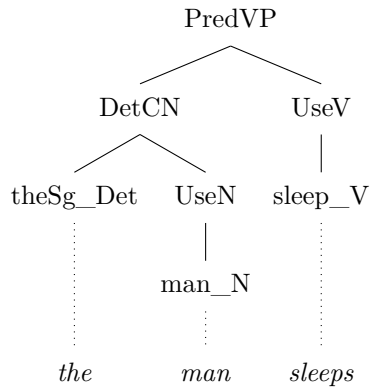
$$
\begin{aligned}
&(\texttt{PredVP} \wedge \texttt{DetCN} \wedge \texttt{UseV} \wedge \texttt{UseN} \wedge \texttt{theSg\_Det} \wedge \texttt{man\_N} \wedge \texttt{sleep\_V}) \\
\vee\ &(\texttt{PredVP\#DetCN\#?} \wedge \texttt{theSg\_Det} \wedge \texttt{UseN\#man\_N} \wedge \texttt{UseV\#sleep\_V}) \\
\vee\ &(\texttt{PredVP\#?\#UseV} \wedge \texttt{DetCN\#theSg\_Det\#?} \wedge \texttt{UseN\#man\_N} \wedge \texttt{sleep\_V}) \\
\vee\ &\ldots
\end{aligned}
$$

The labels for the subtrees are depth-first concatenations of the subtree nodes using the delimiter "#" between the function names, and the question mark character "?" to mark the nodes where the tree has been split. After solving the constraint problem we can either recover the rules from the subtrees in the solution or we can merge the subtrees into new grammar rules.

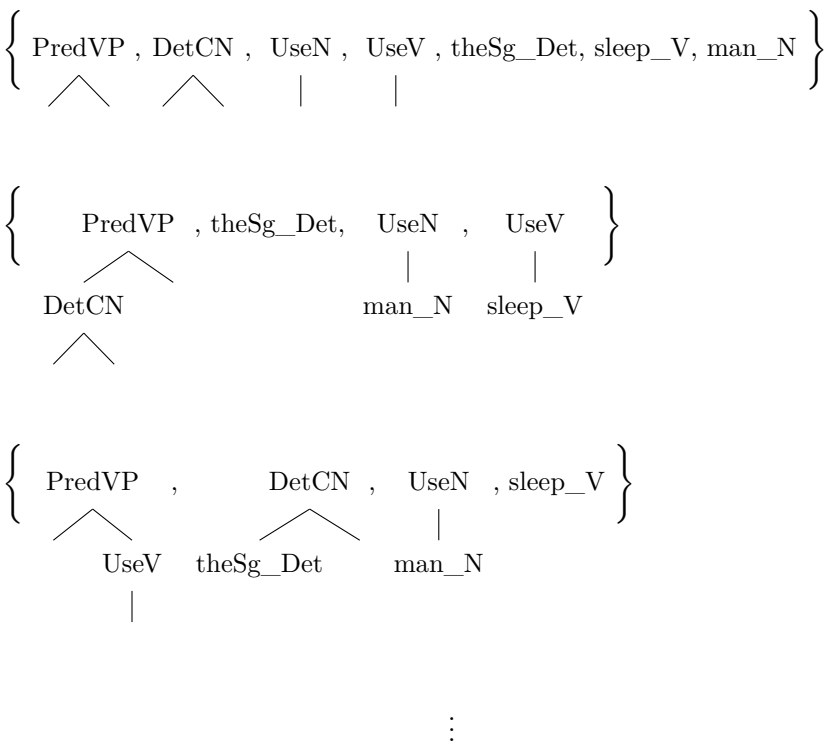
9.9.1 Handling Combinatorial Explosion

The previous section shows that we can easily extend the basic technique to include subtrees or arbitrary size. However, a major challenge is the exponential explosion. If we include all splits into subtrees up to the maximum size of 2 for the tree in Figure 9.19, we already end up with 19 splits and a total of 133 subtrees and if we include subtrees up to a size of 3 we end up with 40 splits and 280 subtrees.

One way to tackle the problem is to limit the number of subtrees with size larger than 1 per split. As we have just seen, if we allow any split into subtrees up to a maximum size, we get a large number of splits, quickly growing with the maximum size of subtrees. When limiting the number of subtrees we can significantly reduce this number of splits. For example, if we would allow at most one subtree of a size up to 2, for the example in Figure 9.19, the first split into only subtrees of size 1 would be allowed. The other two splits given as an example would be, however, disallowed. After starting with a subtree of size 2, only subtrees of size 1 would be allowed for the rest of the split. This solves some of the combinatorial problems but leads to the introduction of additional parameters that influence the learning process.



(a) Original tree



(b) Splitting into subtrees

Figure 9.19: Splitting into subtrees for the example tree (Figure 9.3)

9.9.2 Examples

With the addition of using subtrees to the learning process, we can revisit the Dyck language and the handling of adverbials. We show how the learning method profits from using subtrees and merging rules to solve the problems in a more powerful and elegant way.

9.9.2.1 Dyck Language

In Section 9.8.1.1 we presented the Dyck language with two types of brackets. We demonstrated how the Dyck language can be learned using positive and negative examples. This was possible because of the structure of the grammar we defined. With a different definition of the base language the previous technique cannot learn the desired grammar just using positive and negative examples.

We can define this alternative grammar in Figure 9.20. With this grammar we cannot just exclude rules to learn the intended language. Instead we need to merge the wrap rule with either the rules to add parentheses or with the rules to introduce square brackets.

To learn the grammar we used only positive examples and a moderate subtree size. The examples involved were just the two strings "[( )]" and "[ ] ( )", a subtree size of 3 and allowed at most 2 subtrees in each split. To guarantee for an optimal grammar we used the objective function minimizing the number of rules. The resulting grammar is shown in Figure 9.21.

9.9.2.2 Adverbials

In a similar way we can return to adverbials. Here a major remaining problem is the fact that various phrases are mapped onto the syntactic category of adverbials (`Adv`). And not all phrases make sense in every position an adverbial can appear. For example, prepositional phrases can both modify verbs and nouns. This leads to well-known cases of prepositional phrase (PP) attachment ambiguity. Sometimes the different readings are equally plausible,[8] but usually there is one preferred reading, which can be inferred from the lexical semantics of the involved words.

To give an example, the sentence *I eat pizza with pineapple* is structurally ambiguous but the meaning is completely clear to a human. The same is true for the sentence *I eat pizza with scissors*, even though it would be possible to put scissors on top of a pizza.[9] So the main difference between the two sentences is more an aspect of semantics. In the first case clearly the noun is modified by the prepositional phrase while in the second case it is the verb phrase that is modified.

One potential learning task is to learn a grammar that only allows prepositional phrases modifying nouns but allowing for regular adverbs modifying verbs.

[8] e.g., *I saw the building with the telescope* has again two syntactic analyses, and each one has a plausible semantic interpretation

[9] For some people that would even be as likely as pineapple as a topping.

We start from the grammar in Figure 9.22, which again is a subset of the RGL. With this grammar and the correct examples, the technique can learn a new grammar that disambiguates the attachment ambiguity by allowing only the modification of noun phrases by prepositional phrases.

To learn the new grammar, we use the first sentence as a positive example and the second sentence as a negative example. We add additional sentences to the positive examples to reinforce the prepositional attachment to the noun and to allow for regular verbs modifying verbs.

The positive examples we use for training are:

- *I eat pizza with pineapple*
- *pizza with pineapple is delicious*
- *I run today*
- *I sleep now*
- *I run*

And the only negative example is:

- **I eat pizza with scissors*

We combine these examples with the following training parameters: maximum subtree size of 2 and at most 3 merges per split. As a result we end up with a new grammar that fulfills our expectation about the prepositional attachment. The resulting grammar rules are given in Figure 9.23.

The first rule (`AdvNP#?#PrepNP`) allows prepositional phrases to modify noun phrases. The subsequent two rules allow the two regular adverbs to modify verb phrases. This grammar meets our expectation about the behavior of adverbial. On the other hand it is in some cases overly-specific and in other cases more general than expected. The second (`AdvVP#?#now_Adv`) and third rule (`AdvVP#?#today_Adv`) could be split to make the grammar more general and the rules `UseN` and `MassNP` could be merged because they are the only two rules with matching types. These issues could probably be solved by fine-tuning the parameters. Despite that, this example shows how learning from subtrees and merging to form new grammar rules can be used to deal with a common attachment ambiguity problem.

```
signature Σ_Dyck2 ≡
  sorts
    Dyck, Open, Close
  functions
    Empty : → Dyck
    LeftP, LeftS : → Open
    RightP, RightS : → Close
    Wrap : Open, Dyck, Close → Dyck
    Conc : Dyck, → Dyck
endsignature
```

Figure 9.20: Second over-generating grammar for the Dyck language

```
signature Σ_Dyck2' ≡
  sort Dyck
  functions
    Empty                : → Dyck
    Wrap#LeftP#?#RightP  : Dyck → Dyck
    Wrap#LeftS#?#RightS  : Dyck → Dyck
    Conc                 : Dyck, Dyck → Dyck
endsignature
```

Figure 9.21: Resulting grammar for the Dyck language

```
signature Σ_Adverb ≡
  sorts
    A, Adv, AP, Cl, CN, Comp, NP, Prep, Pron, V, V2, VPSlash
  functions
    AdvNP : NP, Adv → NP
    AdvVP : VP, Adv → VP
    CompAP : AP → Comp
    ComplSlash : VPSlash, NP → VP
    MassNP : CN → NP
    PositA : A → AP
    PredVP : NP, VP → Cl
    PrepNP : Prep, NP → Adv
    SlashV2a : V2 → VPSlash
    UseComp : Comp → VP
    UseN : N → CN
    UsePron : Pron → NP
    UseV : V → VP
    delicious_A : → A
    now_Adv, today_Adv : → Adv
    cheese_N, pineapple_N, pizza_N, scissors_N : → N
    with_Prep : → Prep
    I_Pron : → Pron
    run_V, sleep_V : → V
    eat_V2 : → V2
endsignature
```

Figure 9.22: Signature of a RGL fragment exposing PP attachment ambiguity

```
signature Σ_Adverb ≡
  sorts
    Adv, AP, Cl, CN, NP, Prep, V2
  functions
    AdvNP#?#PrepNP : NP, Prep, NP → NP
    AdvVP#?#now_Adv, AdvVP#?#today_Adv : VP → VP
    ComplSlash#SlashV2a : V2, NP → VP
    MassNP : CN → NP
    PositA#delicious_A : A → AP
    PredVP : NP, VP → Cl
    UseComp#CompAP : AP → VP
    UseN : N → CN
    UsePron#I_Pron : → NP
    UseV#run_V, UseV#sleep_V : V → VP
    pineapple_N, pizza_N : → N
    with_Prep : → Prep
    eat_V2 : → V2
endsignature
```

Figure 9.23: Resulting grammar rules for adverbials including merged rules

9.10 Discussion

In the previous sections we described the technical details of our system. Furthermore, we describe two extensions of the basic technique. All these aspects are implemented and can be tested and evaluated. However, the work described here is only the beginning of an interesting line of research and there are still topics open for discussion.

Influence and Handling of Parameters

There are a few open issues involving the choice and effect of the parameters such as subtree size and how to split trees into subtrees. A serious consequence of starting from the wrong parameters is, that it makes it difficult to learn the intended grammar in the iterative process sketched above.

Some of the parameters lead to the process becoming too slow to be feasible. The number of variables involved in the process, especially in the objective function, slow down the solving of the problem. The objective function reducing the number of trees is usually unproblematic but the objective function minimizing the number of rules can lead to serious problems. This is especially the case when including subtrees because each distinct subtree will be treated as a separate rule.

Another issue is a consequence of restricting the number of subtrees included in a split in combination with negative examples. Because positive and negative examples are treated slightly differently, it can happen, that positive and negative trees are split differently. This means that not all parts of a negative example can be used to eliminate solutions.

These observations are not overly surprising. The more complex a system grows, the more parameters can be tuned. And tuning parameters has a growing effect on the results. In our case, a way to approach this problem is

to start from the most simple system, in our case learning from only positive examples and only add more features when necessary. Another approach is to automatically and iteratively increase the parameter values until a suitable solution can be found.

Handling Larger Problem Sizes

With the basic learning algorithm we did not encounter any performance issues, even though the problem itself is NP-complete. However, one potential problem is the number of parse trees involved, which can grow exponentially in the length of the sentences (Ljunglöf, 2004, p. 7). If we also split the trees into all possible subtrees, the number grows even more. We currently solve this problem by limiting the number of subtrees, but there are other ways to approach this problem as well.

One possible solution is to move away from the formulation of the problem of grammar learning, as covered in this chapter, in terms of parse trees and instead refer to the states in the parse chart. The chart has a polynomial size (Ljunglöf, 2004, p. 87), compared to the exponential growth of the trees, and it should be possible to translate the chart directly to a complex logical formula instead of having to go via parse trees.

Another approach is to use a different constraint solving method. Instead of modeling a constraint optimization problem that requires more effort to solve we can model it as a plain constraint satisfaction problem such as Boolean satisfiability (SAT). This saves us the additional effort in solving a more challenging problem and in translating between logic formulas and ILP constraints, but we lose the guarantee of an optimal solution. However, there are methods to approximate optimal solutions using off-the-shelf SAT solvers, e.g., MiniSAT (Eén and Sörensson, 2003) or SAT+ (Claessen, 2018).

Interaction Between Iterative Process and Merging Rules

Another open question is how exactly the merging of rules can be included in the iterative grammar generation process. To also take advantage of learning from subtrees, it is also possible to include, in each learning step, singular subtrees to occasionally merge rules in cases where the rule-based learning method is not sufficiently powerful.

How well this works in practice is not yet established. Our intuition is that merging rules in a meaningful way requires additional user interaction besides judging positive and negative examples because merging rules could make a grammar more restrictive than intended and might have to be rolled back.

Multilingual Learning

Finally, a very interesting topic we could only touch on shortly is the influence of combining languages in bilingual or multilingual learning. In the preliminary results of the modified “Compare-Against-Treebank” experiment (Section 9.7.2) and an example from the used treebank (Section 9.4), we could show that Finnish and English can be paired up in a meaningful way to disambiguate

features of both languages. However, we did not research the influence of the choice of languages involved more thoroughly.

Another interesting aspect of pairing languages is when encountering lexical ambiguity. The same kind of lexical ambiguity can span across languages, e.g., *bank* is ambiguous in many Germanic languages, not always with the same meanings. But in many cases it is possible to disambiguate the meaning of words by using translations.

9.11 Conclusion

In this chapter we have shown how it is possible for a computer to learn an application- or domain-specific grammar from a very limited number of example sentences. When making use of a large-scale resource grammar, in most cases only around 10 example sentences are enough to get a suitable domain-specific grammar.

We evaluated this method in two different ways with good results that encouraged us to work on the extensions.

Based on the results of the initial method, we also present two modifications. The first one, including negative examples, gives an easy way for humans to influence the learned grammar by giving both sentences that should and should not be included. The second, learning from subtrees and merging rules, allows for more fine-grained domain-specific grammars.

We demonstrated that the procedure developed by our method can learn interesting languages or features using even fewer positive and negative example sentences on two examples involving both formal languages and natural language phenomena. Already five sentences were often sufficient to achieve the intended result.

In Section 9.10 we discussed some of the remaining issues of this method. Notwithstanding, we accomplished to present a framework that can be used for human-centric, iterative grammar learning for domain- and application-specific grammars. There is still work left to be done, including performing more evaluations on different kinds of grammars and example treebanks. But we hope that this idea can find its uses in areas such as computer-assisted language learning, domain-specific dialogue systems, computer games, and more. In our future work, We will especially focus on ways to use this method in computer-assisted language learning. However, a thorough evaluation of the suitability of the extracted grammars has to be conducted for each of these applications and remains as future work.

Furthermore, we plan to explore the use of SAT to model the grammar learning problem. This should help to avoid performance issues but requires a redesign of the whole process to approximate an optimal solution.

Finally, we want to include the iterative learning process in a computer-assisted language learning application and evaluate it thoroughly, both with students and language teachers.

Acknowledgement

We want to thank Krasimir Angelov and Thierry Coquand for pointing us in the direction of many-sorted algebras as a means of formalizing abstract grammars.
This chapter is an extended version of the article (Lange and Ljunglöf, 2020) presented at the Special Session NLPinAI at ICAART 2020.
The work reported in this chapter was supported by the Swedish Research Council, project 2014-04788 (MUSTE: Multimodal semantic text editing).

Bibliography

Abolahrar, Elnaz (2011). "Multilingual Grammar-based Language Training: Computational Methods and Tools". MA thesis. Gothenburg, Sweden: Chalmers University of Technology. URL: https://hdl.handle.net/20.500.12380/148140.

Alfter, David and Elena Volodina (2018). "Towards Single Word Lexical Complexity Prediction". In: *Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2018)*. New Orleans, Louisiana: Association for Computational Linguistics (ACL), pp. 79–88. DOI: 10.18653/v1/W18-0508.

Angelov, Krasimir and Aarne Ranta (2010). "Implementing Controlled Languages in GF". In: *Proceedings of the 2009 Conference on Controlled Natural Language (CNL 2009)*. Marettimo Island, Italy: Springer, pp. 82–101. URL: http://dl.acm.org/citation.cfm?id=1893475.1893482.

Baldi, Philip and Pierluigi Cuzzolin (2009). *New perspectives on historical Latin syntax. 1, Syntax of the sentence*. eng. Trends in linguistics. Studies and monographs, 180.1. Berlin ; New York: Mouton de Gruyter. ISBN: 9783110190823.

— (2010a). *New perspectives on historical Latin syntax. 2, Constituent Syntax ; Adverbial Phrases, Adverbs, Mood, Tense*. eng. Trends in linguistics. Studies and monographs, 180.2. Berlin ; New York: Mouton de Gruyter. ISBN: 9783110205633.

— (2010b). *New perspectives on historical Latin syntax. 3, Constituent Syntax ; Quantification, Numerals, Possessions, Anaphora*. eng. Trends in linguistics. Studies and monographs, 180.3. Berlin ; New York: Mouton de Gruyter. ISBN: 9783110207545.

— (2011). *New perspectives on historical Latin syntax. 4, Complex Sentences, Grammaticalization, Typology*. Trends in linguistics. Studies and monographs, 180.4. Berlin ; New York: Mouton de Gruyter. ISBN: 9783110253405.

Bamman, David and Gregory Crane (2006). "The Design and Use of a Latin Dependency Treebank". In: *Proceedings of the Fifth International Treebanks and Linguistic Theories Conference*. Ed. by Jan Hajic and Joakim Nivre. Institute of Formal, Applied Linguistics, Faculty of Mathematics, and Physics, Charles University. Prag, pp. 67–78. URL: http://hdl.handle.net/10427/42684.

Baños Baños, José M., ed. (2011). *Sintaxis del latín clásico*. Madrid, Spain: Liceus. ISBN: 9788498228441.

Bax, Stephen (2003). "CALL — Past, Present and Future". In: *System* 31.1, pp. 13–28. DOI: 10.1016/S0346-251X(02)00071-4.

Bayer, Karl and Josef Lindauer, eds. (1994). *Lateinische Grammatik*. 2. Edition, auf der Grundlage der Lateinischen Schulgrammatik von Landgraf-Leitschuh neu bearbeitete. Bamberg and Munich: C.C. Buchners Verlag, J. Lindauer Verlag, and R. Oldenburg Verlag.

Bod, Rens (1992). "A Computational Model of Language Performance: Data Oriented Parsing". In: *Proceedings of the 14th International Conference on Computational Linguistics (COLING 1992)*. Nantes, France: Association for Computational Linguistics (ACL). URL: https://www.aclweb.org/anthology/papers/C/C92/C92-3126/.

Bod, Rens and Remko Scha (1997). "Data-Oriented Language Processing". In: *Corpus-based Methods in Language and Speech Processing*. Ed. by Steve Young and Gerrit Bloothooft. Text, Speech, and Language Technology 2. Dordrecht: Springer. Chap. 5, pp. 137–174. DOI: 10.1007/978-94-017-1183-8_5.

Bresnan, Joan (2001). *Lexical-Functional Syntax*. Blackwell Textbooks in Linguistics. Malden, Massachusetts: Blackwell. ISBN: 0631209735.

Bryman, Alan (2012). *Social Research Methods*. 4th edition. Great Clarendon Street, Oxford, UK: Oxford University Press.

Caesar, C. Julius (1869). *Caesar's Gallic War*. Trans. by W. A. McDevitte and W. S. Bohn. 1st. Harper's New Classical Library. New York, NY, USA: Harper & Brothers.

Chen, Danqi and Christopher D. Manning (2014). "A Fast and Accurate Dependency Parser using Neural Networks". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP'14)*. Doha, Qatar: Association for Computational Linguistics (ACL), pp. 740–750. DOI: 10.3115/v1/D14-1082.

Chomsky, Noam (1957). *Syntactic Structures*. Reprint 2002. Berlin and New York: Mouton de Gruyter.

Claessen, Koen (2018). *SAT+*. https://github.com/koengit/satplus. accessed 25-June-2020.

Clark, Alexander (2000). "Inducing Syntactic Categories by Context Distribution Clustering". In: *Proceedings of the 4th Conference on Computational Natural Language Learning (CoNNL 2000)*. Lisbon, Portugal: Association for Computational Linguistics (ACL), pp. 91–94. URL: https://www.aclweb.org/anthology/W00-0717/.

— (2001). "Unsupervised Induction of Stochastic Context Free Grammars with Distributional Clustering". In: *Proceedings of 5th Conference on Computational Natural Language Learning (CoNNL 2001)*. Toulouse, France: Association for Computing Machinery (ACM), pp. 105–112. DOI: 10.3115/1117822.1117831.

Clark, Alexander and Shalom Lappin (2010). "Unsupervised Learning and Grammar Induction". In: *The Handbook of Computational Linguistics and Natural Language Processing*. Ed. by Alexander Clark, Chris Fox, and Shalom Lappin. Oxford, UK: Wiley-Blackwell. Chap. 8, pp. 197–220. DOI: 10.1002/9781444324044.ch8.

Clark, Alexander and Ryo Yoshinaka (2014). "Distributional Learning of Parallel Multiple Context-free Grammars". In: *Machine Learning* 96.1–2, pp. 5–31. DOI: 10.1007/s10994-013-5403-2.

Computer History Museum (2010). *PLATO@50: PLATO Computer Learning System 50th Anniversary*. https://www.youtube.com/watch?v=THoxsBw-UmM. accessed 08-June-2018.

Cormen, Thomas H (2009). *Introduction to Algorithms*. 3rd edition. The MIT Press. Cambridge, MA, USA. ISBN: 9780262270830.

Crane, Gregory R., ed. (2018). *Perseus Digital Library*. http://www.perseus.tufts.edu. accessed 09-June-2018.

Csikszentmihalyi, Mihaly (1990). *Flow: The Psychology of Optimal Experience*. New York, NY, USA: Harper & Row.

Curry, Haskell B. (1961). "Some Logical Aspects of Grammatical Structure". In: *Structure of Language and its Mathematical Aspects. Proceedings of Symposia in Applied Mathematics*. Ed. by Roman Jakobson. Vol. 12. New York, NY, USA: American Mathematical Society.

D'Ulizia, Arianna, Fernando Ferri, and Patrizia Grifoni (2011). "A Survey of Grammatical Inference Methods for Natural Language Learning". In: *Aritifical Intelligence Review* 36, pp. 1–27. DOI: 10.1007/s10462-010-9199-1.

DELPH-IN (2020). *Deep Linguistic Processing with HPSG (DELPH-IN)*. http://moin.delph-in.net/GrammarCatalogue. accessed 25-June-2020.

Deterding, Sebastian, Dan Dixon, Rilla Khaled, and Lennart Nacke (2011). "From Game Design Elements to Gamefulness: Defining 'Gamification'". In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*. Tampere, Finland: Association for Computing Machinery (ACM), pp. 9–15. DOI: 10.1145/2181037.2181040.

Détrez, Grégoire and Aarne Ranta (2012). "Smart Paradigms and the Predictability and Complexity of Inflectional Morphology". In: *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL'12)*. Avignon, France: Association for Computing Machinery (ACM), pp. 645–653. URL: http://dl.acm.org/citation.cfm?id=2380816.2380895.

Dimitrijevic, Dragana (2017). "Latin Curricula, Attitudes and Achievement: An Empirical Investigation". In: *Proceedings of the 19th International Colloquium on Latin Linguistics (ICLL 2017)*. Munich, Germany.

Eckhoff, Hanne M., Kristin Bech, Gerlof Bouma, Kristine Gunn Eide, Dag T. Haug, Odd Einar Haugen, and Marius L. Jøhndal (2018). "The PROIEL Treebank Family: A Standard for Early Attestations of Indo-European Languages". In: *Language Resources and Evaluation* 52.1, pp. 29–65. DOI: 10.1007/s10579-017-9388-5.

Eén, Niklas and Niklas Sörensson (2003). "An Extensible SAT-solver". In: *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*. Portofino, Italy: Springer, pp. 502–518. DOI: 10.1007/978-3-540-24605-3_37.

Ehrling, Sara (2015). *Lingua Latina novo modo – En nybörjarbok i latin för universitetsbruk*. Gothenburg, Sweden: University of Gothenburg.

Felzer, Torsten, Ian Scott MacKenzie, and Stephan Rinderknecht (2014). "Efficient Computer Operation for Users with a Neuromuscular Disease with OnScreenDualScribe". In: *Journal of Interaction Science* 2.2. DOI: 10.1186/s40166-014-0002-7.

Forsberg, Markus and Aarne Ranta (2004). "Functional Morphology". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Functional Programming (ICFP 2004)*. Snow Bird, UT, USA: Association for Computing Machinery (ACM), pp. 213–223. DOI: 10.1145/1016850.1016879.

Fort, Karën, Bruno Guillaume, and Hadrien Chastant (2014). "Creating Zombilingo, a Game with a Purpose for Dependency Syntax Annotation". In: *Proceedings of the 1st International Workshop on Gamification for Information Retrieval (GamifIR '14)*. Amsterdam, The Netherlands: Association for Computing Machinery (ACM), pp. 2–6. DOI: 10.1145/2594776.2594777.

Fox-Turnbull, Wendy (2009). "Stimulated Recall using Autophotography. A Method for Investigating Technology Education". In: *Proceedings PATT-22 Conference. Strengthening the Position of Technology Education in the Curriculum*, pp. 204–217. URL: https://www.iteea.org/File.aspx?id=86963&v=46b05ce9.

Fuchs, Norbert E., Stefan Höfler, Kaarel Kaljurand, Fabio Rinaldi, and Gerold Schneider (2005). "Attempto Controlled English: A Knowledge Representation Language Readable by Humans and Machines". In: *Reasoning Web, 1th International Summer School 2005*. Ed. by Norbert Eisinger and Jan Małuszyński. Lecture Notes in Computer Science 3564. Msida, Malta: Springer. DOI: 10.1007/11526988_6.

Garcia, Ignacio (2013). "Learning a Language for Free while Translating the Web. Does Duolingo Work?" In: *International Journal of English Linguistics* 3.1, p. 19. DOI: 10.5539/ijel.v3n1p19.

Garey, Michael R. and David S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co. ISBN: 0-7167-1045-5.

Gazdar, Gerald (1985). *Generalized Phrase Structure Grammar*. Oxford, UK: Blackwell. ISBN: 0631132066.

Glück, Helmut, ed. (2004). *Metzler Lexikon Sprache*. 2nd edition. Digitale Bibliothek 34. Berlin, Germany: Directmedia.

Gruzitis, Normunds and Dana Dannélls (2017). "A Multilingual FrameNet-based Grammar and Lexicon for Controlled Natural Language". In: *Language Resources and Evaluation* 51.1, pp. 37–66. DOI: 10.1007/s10579-015-9321-8.

Gwynn, Aubrey (1926). *Roman Education from Cicero to Quintilian*. Oxford, UK: Clarendon Press.

Hallett, Catalina, Donia Scott, and Richard Power (2007). "Composing Questions through Conceptual Authoring". In: *Computational linguistics* 33.1, pp. 105–133. DOI: 10.1162/coli.2007.33.1.105.

Henschel, Renate (1997). "Application-Driven Automatic Subgrammar Extraction". In: *Computational Environments for Grammar Development and Linguistic Engineering*. Stroudsburg, PA, USA: Association for Computational Linguistics (ACL). URL: https://www.aclweb.org/anthology/W97-1507.

Imada, Keita and Katsuhiko Nakamura (2009). "Learning Context Free Grammars by Using SAT Solvers". In: *Proceedings of the 8th International Conference on Machine Learning and Applications (ICMLA 2009)*. Miami Beach, FL, USA: IEEE, pp. 267–272. DOI: 10.1109/ICMLA.2009.28.

Joshi, Aravind K. and Yves Schabes (1997). "Tree-Adjoining Grammars". In: *Handbook of Formal Languages: Volume 3 Beyond Words*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Berlin and Heidelberg: Springer, pp. 69–123. DOI: 10.1007/978-3-642-59126-6_2.

Kamp, Hans and Uwe Reyle (1993). *From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Vol. 42. Studies in Linguistics and Philosophy. Dordrecht: Springer. ISBN: 978-0-7923-1028-0. DOI: 10.1007/978-94-017-1616-1.

Kaplan, Ronald M. and Joan Bresnan (1982). "Lexical-Functional Grammar: A Formal System for Grammatical Representations". In: *The Mental Representation of Grammatical Relations* 47, pp. 173–281.

Karp, Richard M. (1972). "Reducibility Among Combinatorial Problems". In: *Complexity of Computer Computations*. Ed. by R. E. Miller, J. W. Thatcher, and J.D. Bohlinger. New York, NY, USA: Plenum, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

Kaya, Hasan and Gülşen Eryiğit (2015). "Using Finite State Transducers for Helping Foreign Language Learning". In: *Proceedings of the 2nd Workshop on Natural Language Processing Techniques for Educational Applications (BEA 2015)*. Stroudsburg, PA, USA: Association for Computational Linguistics (ACL), pp. 94–98. DOI: 10.18653/v1/W15-4414.

Kenji Horie, André (2017). *Rewriting Duolingo's Engine in Scala*. http://making.duolingo.com/rewriting-duolingos-engine-in-scala. accessed 04-April-2018.

Kešelj, Vlado and Nick Cercone (2007). "A Formal Approach to Subgrammar Extraction for NLP". In: *Mathematical and Computer Modelling* 45.3, pp. 394–403. DOI: 10.1016/j.mcm.2006.06.001.

Kilgarriff, Adam, Miloš Husák, Katy McAdam, Michael Rundell, and Pavel Rychlý (2008). "GDEX: Automatically Finding Good Dictionary Examples in a Corpus". In: *Proceedings of the 13th EURALEX International Congress*. Barcelona, Spain: EURALEX, pp. 425–432. URL: https://euralex.org/publications/gdex-automatically-finding-good-dictionary-examples-in-a-corpus/.

Kolachina, Prasanth (2019). "Multilingual Abstractions: Abstract Syntax Trees and Universal Dependencies". PhD thesis. Gothenburg, Sweden: University of Gothenburg. URL: http://hdl.handle.net/2077/60331.

Kolachina, Prasanth and Aarne Ranta (2016). "From Abstract Syntax to Universal Dependencies". In: *Linguistic Issues in Language Technology (LiLT)*. Vol. 13. 3. CSLI, Stanford.

Kuhn, Tobias (2014). "A Survey and Classification of Controlled Natural Languages". In: *Computational Linguistics* 40.1, pp. 121–170. DOI: 10.1162/COLI_a_00168.

Kuhn, Tobias and Rolf Schwitter (2008). "Writing Support for Controlled Natural Languages". In: *Proceedings of the Australasian Language Technology Association Workshop 2008*, pp. 46–54. URL: https://www.aclweb.org/anthology/U08-1007/.

Kumar, Anuj, Tim Paek, and Bongshin Lee (2012). "Voice Typing: A New Speech Interaction Model for Dictation on Touchscreen Devices". In: *Proceedings of CHI 2012, SIGCHI Conference on Human Factors in Computing Systems*. Austin, Texas: Association for Computing Machinery (ACM). DOI: 10.1145/2207676.2208386.

Lange, Herbert (2013). "Erstellung einer Grammatik für das Lateinische im 'Grammatical Framework'". MA thesis. Munich, Germany: Ludwig-Maximilians-Universität.

— (2017). "Implementation of a Latin Grammar in Grammatical Framework". In: *Proceedings of the 2nd International Conference on Digital Access to Textual Cultural Heritage (DATeCH 2017)*. Göttingen, Germany: Association for Computing Machinery (ACM), pp. 97–102. DOI: 10.1145/3078081.3078108.

— (2018). "Computer-Assisted Language Learning with Grammars. A Case Study on Latin Learning". Licentiate thesis. Gothenburg, Sweden: University of Gothenburg. URL: https://gup.ub.gu.se/file/207536.

— (2020). *GF Latin Resource Grammar Evaluation*. DOI: 10.17605/OSF.IO/UWJ59.

Lange, Herbert and Peter Ljunglöf (2018a). "MULLE: A Grammar-based Latin Language Learning Tool to Supplement the Classroom Setting". In: *Proceedings of the 5th Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA 2018)*. Melbourne, Australia: Association for Computational Linguistics (ACL), pp. 108–112. DOI: 10.18653/v1/W18-3715.

— (2018b). "Putting Control into Language Learning". In: *Proceedings of the 6th International Workshop on Controlled Natural Languages (CNL 2018)*. Vol. 304. Frontiers in Artificial Intelligence and Applications. Maynooth, Ireland: IOS Press, pp. 61–70. DOI: 10.3233/978-1-61499-904-1-61.

— (2020). "Learning Domain-Specific Grammars from a Small Number of Examples". In: *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART 2020) - Volume 1: NLPinAI,* Valletta, Malta: SciTePress, pp. 422–430. DOI: 10.5220/0009371304220430.

Lari, Karim and Steve J. Young (1990). "The Estimation of Stochastic Context-free Grammars Using the Inside-Outside Algorithm". In: *Computer Speech & Language* 4.1, pp. 35–56. DOI: 10.1016/0885-2308(90)90022-X.

Lehmann, Paul (1934). "Die Institutio oratoria des Quintilianus im Mittelalter". In: *Philologus* 89.1-4, pp. 353–387. DOI: 10.1524/phil.1934.89.14.353.

Levy, Michael (1997). *Computer-Assisted Language Learning. Context and Conceptualization*. Oxford, UK: Clarendon Paperback. ISBN: 978-0198236313.

Lewis Ph.D., Charlton T. and Charles Short LL.D. (1879). *A Latin Dictionary. Founded on Andrews' edition of Freund's Latin dictionary. Revised, enlarged, and in great part rewritten*. Oxford, UK: Clarendon Press. URL: http://www.perseus.tufts.edu/hopper/text?doc=Perseus:text:1999.04.0059.

Lilliesträom, Ann, Koen Claessen, and Nicholas Smallbone (2019). "Inferring Morphological Rules from Small Examples Using 0/1 Linear Programming". In: *Proceedings of the 22nd Nordic Conference on Computational Linguistics (NoDaLiDa 2019)*. Turku, Finland: Linköping University Electronic Press, pp. 164–174. URL: https://www.aclweb.org/anthology/W19-6118.

Lindauer, Josef, Klaus Westphalen, and Bernd Kreiler (2000). *Roma, Ausgabe C für Bayern, Bd.1*. Bamberg, Germany: C.C. Buchner.

Ljunglöf, Peter (2004). "Expressivity and Complexity of the Grammatical Framework". PhD thesis. Gothenburg, Sweden: University of Gothenburg. URL: http://hdl.handle.net/2077/16377.

— (2011). "Editing Syntax Trees on the Surface". In: *Proceedings of the 18th Nordic Conference of Computational Linguistics (NoDaLiDa 2011)*. Rīga, Latvia: Association for Computational Linguistics (ACL). URL: https://www.aclweb.org/anthology/W11-4619/.

Manning, Christopher D. and Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: The MIT Press. ISBN: 9780262312134.

Mateu, Jaume and Renato Oniga (2017). "Latin Syntax in Fifty Years of Generative Grammar / La sintaxi llatina al llarg de cinquanta anys de Gramàtica Generativa". In: *Catalan Journal of Linguistics* 16, pp. 5–17. DOI: 10.5565/rev/catjl.213.

Michaud, Lisa N. (2008). "King Alfred: A Translation Environment for Learners of Anglo-Saxon English". In: *Proceedings of the Third Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2008)*. Columbus, Ohio: Association for Computational Linguistics (ACL), pp. 19–26. URL: http://www.aclweb.org/anthology/W08-0903.

Moritz, Maria, Barbara Pavlek, Greta Franzini, and Gregory Crane (2016). "Sentence Shortening via Morpho-syntactic Annotated Data in Historical Language Learning". In: *Journal on Computing and Cultural Heritage (JOCCH)* 9.1, pp. 1–9. DOI: 10.1145/2810040.

Müller-Lancé, Johannes (2006). *Latein für Romanisten. Ein Lehr- und Arbeitsbuch*. 1st edition. Tübingen: Gunter Narr Verlag.

Murphy, James J. (1980). "The Teaching of Latin as a Second Language in the 12th Century". In: *Historiographia Linguistica* 7.1-2, pp. 159–175. DOI: 10.1075/hl.7.1-2.12mur.

Nivre, Joakim et al. (2019). *Universal Dependencies 2.4*. URL: http://hdl.handle.net/11234/1-2988.

Ogden, Charles Kay (1930). *Basic English: A General Introduction with Rules and Grammar*. London, UK: Paul Treber.

Passarotti, Marco, Marco Budassi, Eleonora Litta, and Paolo Ruffolo (2017). "The Lemlat 3.0 Package for Morphological Analysis of Latin". In: *Proceedings of the NoDaLiDa 2017 Workshop on Processing Historical Language*. Gothenburg, Sweden: Association for Computational Linguistics (ACL), pp. 24–31. URL: https://www.aclweb.org/anthology/W17-0506/.

Passarotti, Marco, Flavio Massimiliano Cecchini, Greta Franzini, Eleonora Litta, Francesco Mambrini, and Paolo Ruffolo (2019). "LiLa: Linking Latin – A Knowledge Base of Linguistic Resources and NLP Tools for Latin". In:

Proceedings of the 2nd Conference on Language, Data and Knowledge (LDK 2019). Leipzig, Germany: Zenodo. DOI: 10.5281/zenodo.3358550.

Pereira, Fernando and Yves Schabes (1992). "Inside-Outside Reestimation From Partially Bracketed Corpora". In: *30th Annual Meeting of the Association for Computational Linguistics (ACL 1992)*. Newark, Delaware, USA: Association for Computational Linguistics (ACL), pp. 128–135. DOI: 10.3115/981967.981984.

Pilán, Ildikó, Elena Volodina, and Torsten Zesch (2016). "Predicting Proficiency Levels in Learner Writings by Transferring a Linguistic Complexity Model from Expert-written Coursebooks". In: *Proceedings of the 26th International Conference on Computational Linguistics (COLING 2016): Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, pp. 2101–2111. URL: https://www.aclweb.org/anthology/C16-1198.

Pinkster, Harm (1984). *Latijnse syntaxis en semantiek*. Revised 1988 Lateinische Syntax und Semantik, Tübingen: Francke, 1990 Latin Syntax and Semantics, London: Routledge. Amsterdam: Grüner.

— (2015). *The Oxford Latin Syntax. Volume I: The Simple Clause*. Great Clarendon Street, Oxford, UK: Oxford University Press. ISBN: 9780199283613.

Pollard, Carl Jesse (1994). *Head-driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. Chicago: Univ. of Chicago Press. ISBN: 0226674460.

Ranta, Aarne (2009a). "Grammatical Framework: A Multilingual Grammar Formalism". In: *Language and Linguistics Compass* 3.5, pp. 1242–1265.

— (2009b). "The GF Resource Grammar Library". In: *Linguistic Issues in Language Technology* 2(2). URL: https://journals.linguisticsociety.org/elanguage/lilt/article/view/214/158.html.

— (2011). *Grammatical Framework: Programming with Multilingual Grammars*. Stanford: CSLI Publications. ISBN: 978-1575866260.

— (2012). *Implementing Programming Languages. An Introduction to Compilers and Interpreters*. College Publications. ISBN: 978-1848900646.

— (2015). "Constructive Type Theory". In: *The Handbook of Contemporary Semantic Theory*. Ed. by Shalom Lappin and Chris Fox. Second edition. Blackwell Handbooks in Linguistics. John Wiley & Sons, pp. 345–374. ISBN: 9780470670736.

Ranta, Aarne, Krasimir Angelov, Robert Höglind, Christer Axelsson, and Leif Sandsjö (2017). "A Mobile Language Interpreter App for Prehospital/Emergency Care". In: *Medicinteknikdagarna*. Västerås, Sweden. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:hb:diva-13366.

Ranta, Aarne and Prasanth Kolachina (2017). "From Universal Dependencies to Abstract Syntax". In: *Proceedings of the 1st Workshop on Universal Dependencies*. Linköping University Electronic Press, pp. 107–116. URL: https://www.aclweb.org/anthology/W17-0414.

Redkar, Hanumant, Sandhya Singh, Meenakshi Somasundaram, Dhara Gorasia, Malhar Kulkarni, and Pushpak Bhattacharyya (2017). "Hindi Shabdamitra: A WordNet-based E-Learning Tool for Language Learning and Teaching". In: *Proceedings of the 4th Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA 2017)*. Taipei,

Taiwan: Asian Federation of Natural Language Processing, pp. 23–28. URL: http://aclweb.org/anthology/W17-5904.

Reichenbach, Hans (1947). *Elements of Symbolic Logic*. New York, NY, USA: The Macmillan Company.

Russell, Stuart and Peter Norvig (2009). *Artificial Intelligence: A Modern Approach*. 3rd edition. Prentice Hall. ISBN: 978-0136042594.

Sayeed, Asad and Stan Szpakowicz (2004). "Developing a Minimalist Parser for Free Word Order Languages with Discontinuous Constituency". In: *Proceedings of the 4th International Conference on Advances in Natural Language Processing (EsTAL)*. Lecture Notes in Computer Science 3230. Alicante, Spain: Springer. DOI: 10.1007/978-3-540-30228-5_11.

Shibata, Tomoki, Daniel Afergan, Danielle Kong, Beste F. Yuksel, I. Scott MacKenzie, and Robert J.K. Jacob (2016). "Text Entry for Ultra-Small Touchscreens Using a Fixed Cursor and Movable Keyboard". In: *Proceedings of CHI 2016: The ACM SIGCHI Conference Extended Abstracts on Human Factors in Computing Systems*. Santa Clara, California: Association for Computing Machinery (ACM), pp. 3770–3773. DOI: 10.1145/2851581.2890230.

Springmann, Uwe, Helmut Schmid, and Dietmar Najock (2016). "LatMor: A Latin Finite-State Morphology Encoding Vowel Quantity". In: *Open Linguistics* 2.1, pp. 386–392. DOI: 10.1515/opli-2016-0019.

Steedman, Mark (2016). "Combinatory Categorial Grammar. An Introduction". Draft August 25th, 2016.

Stemle, Egon W., Adriane Boyd, Maarten Jansen, Therese Lindström Tiedemann, Nives Mikelić Preradović, Alexandr Rosen, Dan Rosén, and Elena Volodina (2019). "Working Together Towards an Ideal Infrastructure for Language Learner Corpora". In: *Widening the Scope of Learner Corpus Research*. Ed. by Andrea Abel, Aivars Glaznieks, Verena Lyding, and Lionel Nicolas. Corpora and Language in Use. France: Presses universitaires de Louvain. ISBN: 9782875588685.

Swadesh, Morris (1952). "Lexico-Statistic Dating of Prehistoric Ethnic Contacts: With Special Reference to North American Indians and Eskimos". In: *Proceedings of the American Philosophical Society* 96.4, pp. 452–463. URL: http://www.jstor.org/stable/3143802.

— (1955). "Towards Greater Accuracy in Lexicostatistic Dating". In: *International Journal of American Linguistics* 21.2, pp. 121–137. URL: http://www.jstor.org/stable/1263939.

Sweetser, Penelope and Peta Wyeth (2005). "GameFlow: A Model for Evaluating Player Enjoyment in Games". In: *Computers in Entertainment (CIE)* 3.3. DOI: 10.1145/1077246.1077253.

Takahashi, Ayumi and Hideki Takahashi (2015). "Anxiety and Self-confidence in Ancient Language Studies". In: *Niigata University Language and Culture Research Department Bulletin*.

Taylor, Daniel J. (1970). "A Study of the Linguistic Theory of Marcus Terentius Varro". PhD thesis. Seattle, Washington: University of Washington, pp. 1–158.

Taylor, Wilson L. (1953). "'Cloze Procedure': A New Tool for Measuring Readability". In: *Journalism Quarterly* 30.4, pp. 415–433. DOI: 10.1177/107769905303000401.

Vesselinov, Roumen and John Grego (2012). *Duolingo Effectiveness Study*. Tech. rep. accessed 16-October-2017.

Vicipaedia (2018). *Libera Enceclopaedia*. https://la.wikipedia.org/wiki/Vicipaedia:Pagina_prima. accessed 09-July-2018.

— (2020). *Libera Enceclopaedia: Census*. https://la.wikipedia.org/wiki/Specialis:Census. accessed 09-July-2020.

Vilborg, Ebbe (2009). *Norstedts Svensk-Latinska Ordbok*. 2nd edition. Stockholm, Sweden: Norstedts akademiska förlag. ISBN: 9789172275720.

Volodina, Elena, Ildikó Pilán, Lars Borin, and Therese Lindström Tiedemann (2014). "A Flexible Language Learning Platform Based on Language Resources and Web Services". In: *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association (ELRA), pp. 26–31. URL: https://www.aclweb.org/anthology/L14-1684/.

Ward, David J., Alan F. Blackwell, and David J. C. Mackay (2002). "Dasher: A Gesture-Driven Data Entry Interface for Mobile Computing Human-Computer Interaction". In: *Human-Computer Interaction: Text Entry for Mobile Computing* 17.2–3, pp. 199–228. DOI: 10.1080/07370024.2002.9667314.

Warschauer, Mark (2004). "Technological Change and the Future of CALL". In: *New Perspectives on CALL for Second and Foreign Language Classrooms*. Ed. by Sandra Fotos and Charles M. Brown. Mahwah, NJ: Lawrence Erlbaum Associates. Chap. 2, pp. 15–25.

Whitaker, William (2006). *Words. Latin-English Dictionary Program*. http://archives.nd.edu/whitaker/words.htm. accessed 15-July-2018.

Wirsing, Martin (1990). "Algebraic Specification". In: *Handbook of Theoretical Computer Science*. Ed. by Jan van Leeuwen. Amsterdam et al.: Elsevier; The MIT Press. Chap. 13, pp. 677–788.

XMG (2017). *eXtensible MetaGrammar (XMG)*. http://xmg.phil.hhu.de/. accessed 25-June-2020.