Originally published in: DATeCH2017: Proceedings of the 2nd International Conference on Digital Access to Textual Cultural Heritage.
Göttingen, Germany, June 1–2, 2017. – New York: Association for Computing Machinery, 2017. Pp. 97-102.

Herbert Lange
Department for Computer Science and Engineering
University of Gothenburg
412 96, Göteborg, Sweden
herbert.lange@cse.gu.se

Implementation of a Latin Grammar in Grammatical Framework

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.

CCS CONCEPTS

• **Computing methodologies → Language resources**;

KEYWORDS

Grammatical Framework, Latin, grammar development

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") [4].
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 1.1) and the Latin language (Section 1.2). The main part of the work is the description of the implementation of the grammar based on a common Latin grammar book [1]. The implementation (Section 2) consists of a lexicon (Section 2.1), morphology (Section 2.2) and syntax (Section 2.3). The conclusion (Section 3) discusses some future work.

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

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

programming languages in computer science like Haskell to formalize natural languages [6].
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 [6]. 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 Grammars (PMCFG) [3].
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 Listing 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, 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 [5]. It is already implemented for many languages including the Latin grammar presented here.

[2]Tables types have the form $\{Type_1 \Rightarrow Type_2\}$ with $Type_1$ a finite type and $Type_2$ an arbitrary type and values of these types have the form `table {` k_1 `=>` V_1`;...;`k_n `=>` 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. [6, pp. 279-282]

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 [2] 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 [1], 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.

2 IMPLEMENTATION OF THE GRAMMAR

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 [1]. 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.

[3]One interesting late instance of Latin in the field of mathematics is Latino sine Flexione [2]

```
-- Possible Number values: Sg (= Singular), Pl(ural)
-- Possible Case values: Nom(inative), Acc(usative), Dat(ive), Abl(ative) and Voc(ative)
-- Possible Gender values: Masc(uline), Fem(inine), Netr (= Neuter)
-- Type for Nouns:  N = { s : Number => Case => Str ; g : Gender } ;
  lin
    man_N = {
      s = table {
        Sg => table {
          Nom => "vir"; Acc => "virum"; Gen => "viri";
          Dat => "viro"; Abl => "viro"; Voc => "vir" };
        Pl => table {
          Nom => "viri"; Acc => "viros"; Gen => "virorum";
          Dat => "viris"; Abl => "viris"; Voc => "viri"; }
      } ;
      g = Masc
    }
```

Listing 1: An example for a noun in this grammar

the Latin Wikipedia has more than 100.000 pages [8] 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.

2.2 Morphology

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 [6]. An example for a smart paradigm for Latin nouns can be seen in Listing 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.
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 1. The domains of these features are presented in Table 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.

2.2.1 Noun inflection. Nouns in Latin are, as seen in Table 1 declined[6] by case and number while they have an inherent gender. The Latin cases are given in Table 2. To enforce vocative as a separate case is an arguable decision since it usually shares the *nominative singular* form [1, 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 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

[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
[6] Latin noun inflection is called declension

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

Table 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 2: Domains of the finite features

```
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 in 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 2: A smart paradigm for nouns in the grammar

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.

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.

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 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 [7, 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 [1, 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 [1, 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.

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.

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 3 together with the necessary parametric features. The keyword `param` indicates the definition of finite features by listing all possible values. In Listing 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 `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.

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.

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 [2].
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" - 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.

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.

[7] Verb inflection in Latin is called conjugation

```
param
  Agr          = Ag Gender Number Case ;
  VActForm     = VAct VAnter VTense Number Person ;
  Anteriority  = Simul | Anter ;
  Tense        = Pres | Past | Fut | Cond ;
  Polarity     = Pos | Neg ;
  VQForm       = VQTrue | VQFalse ;
  Order        = SVO | VSO | VOS | OSV | OVS | SOV ;
lincat
  AP =  { s : Agr => Str } ;
  CN =  { s : Number => Case => Str ; g : Gender ; preap : AP ; postap : AP } ;
  NP =  { s : Case => Str ; g : Gender ; n : Number ; p : Person } ;
  VP =  { s : VActForm => VQForm => Str ; obj : Str ; adj : Agr => Str } ;
  Cl =  { s : Tense => Anteriority => Polarity => VQForm => Order => Str } ;
  S =   { s : Str } ;
```

Listing 3: Types adjective phrases (APs), common nouns (CNs), noun phrase (NPs), verb phrases (VPs), clauses (Cls) and sentences (S)

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.

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.
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.

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".

[8] in GPSG notation VP/NP, [6] p. 217

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.

REFERENCES

[1] Karl Bayer and Josef Lindauer (Eds.). 1994. *Lateinische Grammatik* (2. Edition, auf der Grundlage der Lateinischen Schulgrammatik von Landgraf-Leitschuh neu bearbeitete ed.). C.C. Buchners Verlag, J. Lindauer Verlag, R. Oldenburg Verlag.
[2] Helmut Glück (Ed.). 2004. *Metzler Lexikon Sprache* (2. ed.). Number 34 in Digitale Bibliothek. Directmedia.
[3] Peter Ljunglöf. 2004. *Expressivity and Complexity of the Grammatical Framework.* Ph.D. Dissertation. Göteborg University.
[4] Johannes Müller-Lancé. 2006. *Latein für Romanisten* (1. ed.). Gunter Narr Verlag.
[5] Aarne Ranta. 2009. The GF Resource Grammar Library. *Linguistic Issues in Language Technology* 2(2) (12 2009). http://elanguage.net/journals/index.php/lilt/article/view/214/158
[6] Aarne Ranta. 2011. *Grammatical Framework.* CSLL Studies in Computational Linguistics.
[7] Hans Reichenbach. 1947. *Elements of Symbolic Logic.* The Macmillan Company.
[8] Vicipaedia. 2016. Vicipaedia Libera Enceclopaedia. https://la.wikipedia.org/wiki/Vicipaedia:Pagina_prima. (2016). Online, accessed 4.4.2016.

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