

Term Paper

|  |
| --- |
| **The Warrior Stabs the Goat: Creating a Dothraki Resource Grammar with Grammatical Framework** |

**Maria Henkel**

2008709 • maria.henkel@hhu.de

**Benno Kuckuck**

2340409 • benno.kuckuck@hhu.de

October 17th, 2016

|  |  |
| --- | --- |
| **Lecturers:** | Yulia Zinova, Rainer Osswald |
| **Course:** | Introduction to "Grammatical Framework"  SS 2016 |

# Contents

[Contents I](#_Toc463710989)

[List of Figures II](#_Toc463710990)

[List of Tables II](#_Toc463710991)

[1. Introduction 1](#_Toc463710992)

[2. The Project 2](#_Toc463710993)

[2.1 About Resource Grammars and the Standard Library 3](#_Toc463710994)

[2.2 Project Structure 5](#_Toc463710995)

[2.3 Implementation of Grammatical Categories in Dothraki 6](#_Toc463710996)

[2.3.1 Nouns and Noun Phrases 6](#_Toc463710997)

[2.3.2 Verbs and Verb Phrases 8](#_Toc463710998)

[2.3.3 (Relative) Clauses and Sentences 11](#_Toc463710999)

[2.3.4 Adjectives 12](#_Toc463711000)

[2.4 Operating Instructions 16](#_Toc463711001)

[3. Discussion 16](#_Toc463711002)

[3.1 Limitations 16](#_Toc463711003)

[3.2 Outlook 16](#_Toc463711004)

[3.3 Conclusion 16](#_Toc463711005)

[Appendix 17](#_Toc463711006)

[Source Code 17](#_Toc463711007)

[References 17](#_Toc463711008)

[Versicherung 19](#_Toc463711009)

## List of Figures

[Figure 1: Mini Resource Grammar Modules and Dependencies (Ranta, 2011a, p. 209) 4](#_Toc463711010)

[Figure 2: Module Structure of a Full Resource Grammar (Ranta, 2011b) 4](#_Toc463711011)

[Figure 3: GF Resource Grammar Library Categories (Grammatical Framework, 2016b) 15](#_Toc463711012)

## List of Tables

[Table 1: Translation Examples 1](#_Toc463711013)

**

|  |
| --- |
| “Khal Drogo shillo chiories she krazaaj. Chiori fin vijazer adraes zoqwe Khales Drogo. Khal Drogo vinde ma chiories ma adraes she krazaaj. Khal Drogo vachrara.” |
| “Khal Drogo met a woman on a mountain. The woman, who protected a turtle, kissed Khal Drogo. Khal Drogo stabbed the woman and the turtle on the mountain. Khal Drogo stinks.“ |
| GF Run-Time System, 2016 |
|  |

|  |
| --- |
|  |

## Introduction

Grammatical Framework (GF) is a functional programming language, created in 1998 (Ranta, 2011a) doubling as a categorial grammar formalism that can be used to implement (natural) language grammars. As GF is “working from a language-independent representation of meaning” (Wikipedia, 2016) it can be used to generate or parse text in different languages, e.g. for use in machine translation.

Since GF uses a symbolic approach to processing and translating language, in most cases its translations are more precise than those of the more commonly used statistical tools (Brown & Frederking, 1995). For example, the “Google Translate” service (Google, 2016) will produce ungrammatical sentences as soon as the input is too complicated while the GF translator (Grammatical Framework, 2016a) has no problems producing a correct translation (see Table 1) in many cases.

Table 1: Translation Examples

|  |  |  |
| --- | --- | --- |
| Input | Output Google Translate | Output GF Translator |
| I heal these women. | Ich diese Frauen zu heilen. | |  |  | | --- | --- | |  | Ich verheile diese Frauen. | |
| The warrior will kiss that cheese behind the tree. | Der Krieger wird Kuss, der Käse hinter dem Baum. | Der Krieger wird jenen Käse hinter dem Baum küssen. |
| Diese heiße Frau heilt diese Königin im Meer. | This hot woman heals the queen of the sea. | This hot woman heals this queen in the sea. |

Symbolic formalisms like GF – unlike Google Translate – depend on a large knowledge base of predefined grammar rules to achieve high-quality language processing: Its own standard library, the GF Resource Grammar Library, covers more than 38 different languages (Ranta, 2015) - although many of them are not yet complete - and is still growing. In 2010, it already consisted of more than 500,000 lines of code (Ranta, 2011a).

As Grammatical Framework is an open-source project (Ranta, 2009) and still under development, everyone is invited to use it or even help further develop it, for example by adding languages to the library, which is the aim of the project described in this paper.

Our goal is to write a GF resource grammar for the Dothraki language and, eventually, be able to add it to the GF Resource Grammar Library. The work described in this paper is meant to lay a foundation for this goal (see Chapter 2).

Dothraki is a constructed language, developed by the language creator and writer David J. Peterson (Peterson, 2016a). The Dothraki are a fictional race of nomadic horse warriors in the “A Song of Ice and Fire” book series by George R. R. Martin (Martin, 2016). As the series was adapted for television by the television network HBO, the language bits found in Martin’s books had to be extended to a fully functional language (Peterson, 2016b): The Dothraki language. Today, it has a vocabulary of over 3,000 words (Tongues of Ice and Fire Wiki, 2016). We will show a few examples of the Dothraki language grammar in this paper, but more information can be found in the book “Living Language Dothraki” (Peterson, 2014), on Peterson’s blog (2016b) and on the Tongues of Ice and Fire Wiki (2016) among many other sources.

This paper aims to serve as a short documentation and explanation of our project. It will cover the project aim and scope, general information regarding the project as well as its structure and explanations of sample functions and the corresponding Dothraki grammar rules (Chapter 2). We will also discuss problems and limitations and briefly describe future plans before we summarize and evaluate the project in its current state and our experience while working on it (Chapter 3).

## The Project

As mentioned before, the aim of this project is to create a resource grammar for the Dothraki language and add it to the GF Resource Grammar Library, so that other Dothraki fans and/or GF enthusiasts may use it. While there are currently a few Dothraki language learning apps and online translators available (e.g. Cognitus Apps, 2016; Fun Translations, 2016), these are very limited, primarily relying on word-for-word translation unaware of any grammar rules. An open-source Dothraki resource grammar for GF, which already provides a wide range of tools and options for integrating with other projects, would undoubtedly be a significant addition to the available range of Dothraki language resources.

### About Resource Grammars and the Standard Library

GF, like other symbolic approaches to machine translation and language processing, is particularly aimed at providing high-quality output in more narrowly focused applications (as opposed to statistical approaches, whose strengths lie more in providing acceptable output in a very wide range of scenarios). Most applications however share a large part of the rules necessary to parse or construct correct sentences, namely those that are concerned with the morphology and syntax of the involved language(s). While these could be implemented ad-hoc for every application, it is much more economic to provide language-specific but application-independent rules as a reusable library. This is the goal of GF’s Resource Grammar Library.

This frees application programmers from having to deal with "specialized linguistic expertise" (Ranta, 2009). By providing a common interface across languages, it becomes considerably easier (in some cases even trivial) for the application programmer to add support for additional languages. Considerable work has been and is being done by the GF developers and resource grammar programmers to make this possible:

*The idea of using a grammar as a software library is, to our knowledge,*

*new in GF. It has required a considerable effort in the design and implementation of the GF programming language: a type system, a module system, compilation techniques, and optimizations. [...] The effort*

*made in this work is supplemented by a substantial effort in the library itself.*

*The code included in the library is more than twice in size compared to the*

*implementation of GF.* (Ranta, 2009)

This division of labour benefits not just the application programmers however, but also the linguistics experts implementing the application-independent libraries: Firstly because a resource grammar once written can “serve an unlimited number of applications” (Ranta, 2011, p. 200), but also because “the existing library specification will help to identify the linguistic issues and avoid pitfalls” by providing a well thought-out, language-independent interface.

### 2.2 Project Goals and Structure

While, in the long-term, our work is aimed towards a full implementation of a Dothraki resource grammar, this was not a realistic goal for this term paper, for two main reasons: First, Ranta estimates that an interval of three to six months of full-time work is needed for an experienced GF programmer to implement a new resource grammar (Ranta, 2011, p. 222), which far exceeds the time frame for this term paper. And secondly, the Dothraki language itself is still a work-in-progress, with significant parts of the grammar either as yet undocumented, or possibly non-existent, which at present makes a full implementation of all the grammatical functions covered by GF’s RGL arguably impossible (for more on this point, see the discussion in Chapter 3).

In Ranta’s book (2011, pp. 237-245, see Figure 1), a mini resource grammar for Italian is implemented as an example.

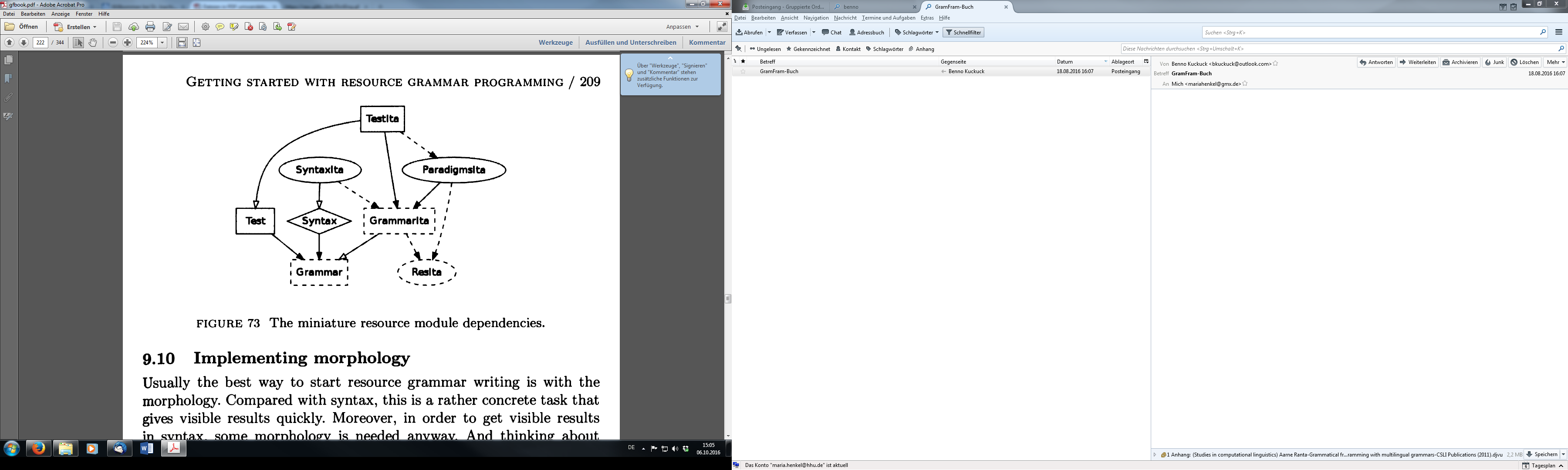


Figure 1: Mini Resource Grammar Modules and Dependencies (Ranta, 2011a, p. 209)

Our work lies somewhere between this example and a full resource grammar. Like in the Italian example, the abstract grammar specification our resource grammar is built on, is only a subset of that of the full RGL. Unlike the Italian example, it adheres much more closely to the structure of the full RGL (see Figure 2), employing the same file structure and API, in hopes of easing a later expansion to a full resource grammar. It is also much more extensive than Ranta’s example, covering a much wider range of grammatical constructions (as a quick comparison, the example from the book implements 12 grammatical categories, whereas our Dothraki implementation covers 27 of the 48 categories of the full RGL; in terms of API methods implemented, the comparison is even starker: Ranta’s example grammar implements 19 API methods, our grammar implements almost 200).

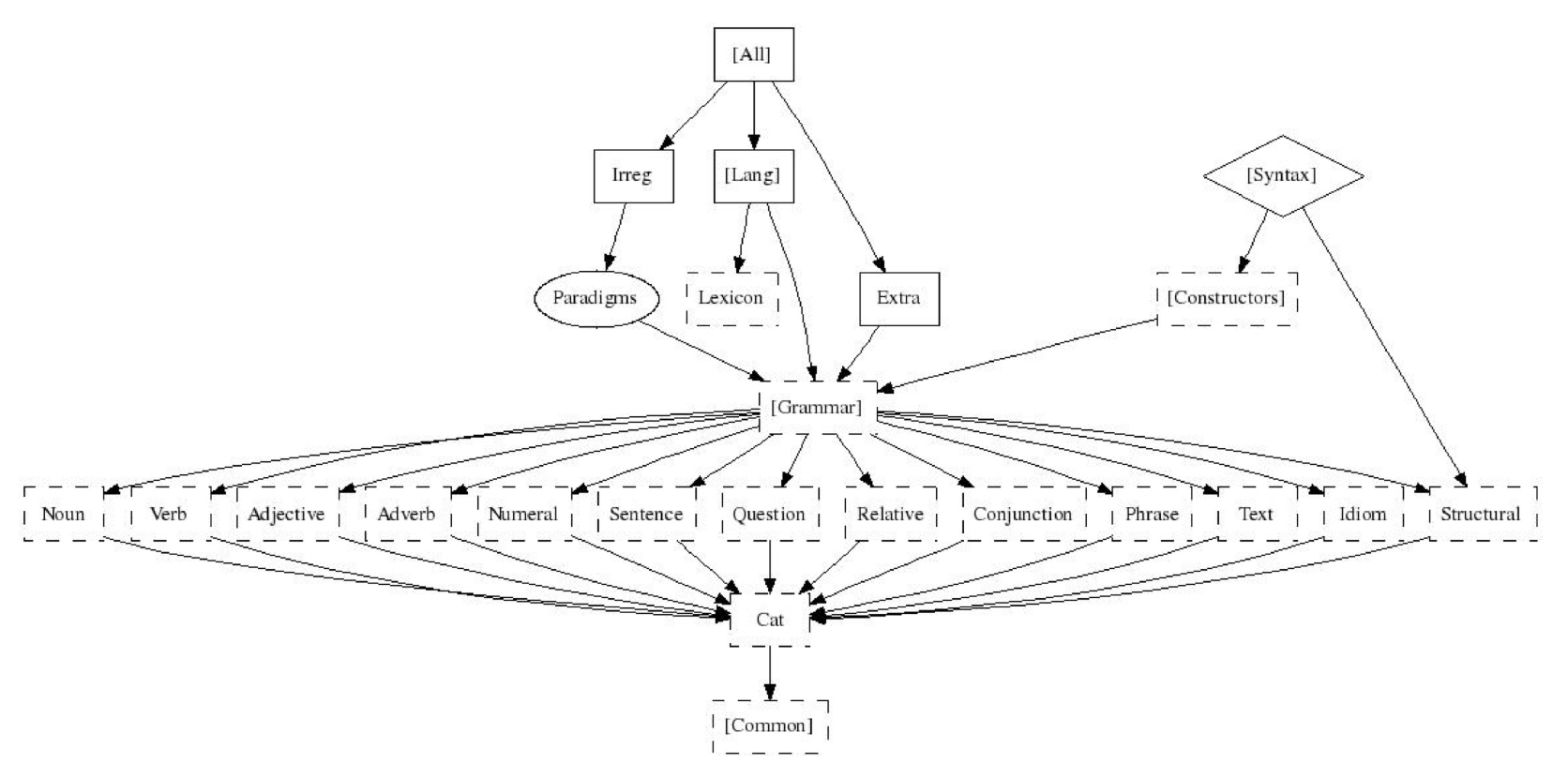


Figure 2: Module Structure of a Full Resource Grammar (Ranta, 2011b)

Just like the GF repositories (Détrez & Camilleri, 2016), our full project is hosted on GitHub (Henkel & Kuckuck, 2016) so that others may see, use and contribute to it. It consists of the following folders:

* simpleapi
* simpleabstract
* simpledothraki
* simpleenglish
* common
* example
* documentation
* vocabulary

The project structure follows closely that of the GF resource grammar library. The “simpleapi” folder (like the “api” folder in the full resource grammar library) contains the outward facing part of the library – the types and functions available to application grammar developers. Those are implemented in terms of more basic functions, which are declared in the abstract grammar(s) contained in the “simpleabstract” folder. For each language, in our case just English and Dothraki, there is one folder (here: “simpledothraki” and “simpleenglish”) which contains the concrete implementations of these abstract grammars. Some of these parts, those shared by all languages, are implemented in the “common” folder. We are planning to remove the “simple” from all folder- and filenames as soon as the resource grammar is complete. Since we already use the structure of a full resource grammar, it should work accordingly. It should be noted, that we are working with the “simpleenglish” resource grammar, to be able to translate Dothraki phrases into English for testing and presenting purposes. Later on, the full English resource grammar, already available via the GF Resource Grammar Library, will be used.

The mini resource grammar can already be used for application grammars. The “example” folder contains application grammar rules to show how the resource grammar can be applied.

There are two more folders: The “documentation” folder is holding two versions of our documentation, one markdown file for readers on GitHub and one print version. Finally, the “vocabulary” folder contains the vocabulary extractor, a supporting python script, written by us to add a relatively large number of words to our dictionary automatically. It takes the source code of the Dothraki vocabulary site (Ice and Fire Wiki, 2016) as input, extracts words and their translations from it and transforms them into the right format, needed for the resource library. For example, the entry "ador [aˈdor], ni. chair" on the website will create the three output strings:

"ador\_N = mkN "ador" inanimate;",

"chair\_N = mkN "chair";" and

"ador\_N = chair\_N;"

One entry for the Dothraki linearization, one entry for the English linearization and one entry for the dictionary. This way, we were able to add more than 1644 words to our dictionary, and will be able to add even more as soon as the resource grammar is expanding.

### Implementation of Grammatical Categories in Dothraki

In this section we will go into some of the design decisions involved in the implementation of the Dothraki resource grammar. In order to get a high-level overview, we will first take a look at the linearizations of some of the categories defined in the API part of the abstract resource grammar library, showing how Dothraki grammar can be fit into the GF scheme.

### Nouns and Noun Phrases

The relevant categories here are ‘N’ (nouns), ‘CN’ (common nouns), ‘NP’ (noun phrases), ‘Pron’ (pronouns) and ‘PN’ (proper names), which are realized as follows:

N, CN = {s : Number => Case => Str ; a : Animacy } ;

NP, Pron = {s : Case => Str ; agr : Agr } ;

PN = {s : Case => Str } ;

In the actual code, some parts of this are defined in the ‘ResDot’ module instead of ‘CatDot’, for easier reusability.

These implementations follow in a fairly straightforward fashion from the basic features of nouns in the Dothraki language: In Dothraki, every noun falls into one of two classes, *animate* and *inanimate* (which play a very similar role to grammatical gender in other languages). The language uses five cases: nominative, accusative, genitive, ablative and allative. Hence, we define:

Animacy = Anim | Inanim ;

Case = Nom | Gen | Acc | All | Abl ;

Noun phrases, as well as pronouns, have a fixed number and person, which is stored in the ‘agr’ field of type ‘Agr’, simply defined as follows:

Agr = Ag Person Number ;

There does not currently seem to be a need to store animacy in the agreement record. This is unlike English, where the ‘NP’s gender can affect the verb phrase ("The boy loves himself." but "The girl loves herself.") and thus needs to be stored in the ‘NP’.

Finally, proper names are (so far as we can tell) always animate and third person singular, so none of that information needs to be stored in ‘PN’.

In order to turn a common noun into a noun phrase, we usually need a determiner. The relevant categories here are ‘Num’ (numerals), ‘Quant’ (quantifiers), ‘Predet’ (predeterminers) and ‘Det’ (determiners, roughly composed of optional predeterminers, a quantifier and a numeral). Numerals in the proper sense, as well as predeterminers, are not yet implemented in our resource grammar (there is a lack of good information on determiner structure in Dothraki), but since the ‘Num’ category is used in the GF Resource Grammar Library not just for representing numerals but also to determine the number of the determiner, we have a dummy implementation in our resource grammar:

Num = {s: Str ; n : Number} ;

Quant = {s : QuForm => Case => Str } ;

Det = {s : Animacy => Case => Str ; n : Number ; s2 : Str } ;

The ‘s’ field of ‘Num’ is always empty in our implementation, this is necessary for technical reasons related to the implementation of GFs parser (Google Groups, 2016).

Quantifiers in Dothraki (such as "jin" 🡪 "this" and "rek" 🡪 "that") are inflected for animacy, number and case, but the singular and plural inanimate forms are always identical, so ‘QuForm’ is defined as follows:

QuForm = QAnim Number | QInanim ;

Unlike a ‘Quant’, a ‘Det’ has a fixed number.

An interesting feature of determiners in Dothraki is that they can appear either before the noun ("jinak adra" 🡪 "this turtle") or after (such as determiners built from possessive pronouns: "adra anni" 🡪 "my turtle"). Conceivably (though we cannot currently say for certain, due to lack of examples and documentation on Dothraki determiner structure), a determiner could even be split into two parts ("my two turtles" 🡪 "akat adrasi anni"?). In anticipation of this, ‘Det’ has two ‘Str’ components, ‘s’ and ‘s2’. Since possessive pronouns are the only examples of postposed determiners we know so far, and these happen not to inflect with animacy or case, the ‘s2’ part is simply a ‘Str’, whereas the ‘s’ part is a table of inflected forms.

### Verbs and Verb Phrases

The relevant categories here are ‘V’ (intransitive verbs), ‘V2’ (transitive verbs with an ‘NP’ complement) and ‘VP’ (verb phrases), whose lincats are as follows:

-- in ResDot --

param VFormPN = Pers1 Number | Pers2 | Pers3 Number ;

param VForm =

APast Polarity Number

| APresent Polarity VFormPN

| AFuture Polarity VFormPN

| ImpFormal Polarity

| ImpInformal Polarity ;

oper Verb : Type = {

s : VForm => Str ;

inf : Str ;

part : Str

} ;

-- in CatSimpleDot --

V = Verb ;

V2 = Verb \*\* {objCase : Case} ;

VPSlash = Verb \*\* {objCase : Case ; subjpost : Str} ;

VP = Verb \*\* {compl : Str; subjpost : Str} ;

The basic structure ‘Verb’ contains the infinitive and participle as well as all conjugated forms of a verb. The somewhat convoluted setup of the ‘VForm’ parameter type is owed to the peculiarities of conjugation in Dothraki: First off, somewhat unusually, in all tenses and moods, verb forms vary with the polarity of the sentence ("Me dothra**e**" 🡪 "He rides", "Me vos dothra**o**" 🡪 "He does not ride"). In the present and future tenses, verbs are inflected for person and number, but the second person singular and plural forms are always identical (hence the definition of ‘VFormPN’). In the past tense, verbs are inflected for number only, not for person (but unlike in the present and future tenses, second person singular and plural forms *do* differ). There are also formal and informal imperative forms.

Transitive verbs are basically the same as intransitive verbs, but they also assign a case to their ‘NP’ complement (mostly accusative, but other cases do appear), which is stored in the ‘objCase’ field.

A verb phrase essentially consists of a verb with a complement. There is also a field ‘subjpost’ which again encodes a peculiar feature of Dothraki grammar: A number of constructions, which, in other languages, are rendered by verbal auxiliaries (can, must) or verbs with ‘VP’ complements (try to), are realized in Dothraki via non-inflecting particles (e.g. "laz" and "eth" corresponding to "can" and "must"), which do not otherwise affect the conjugation of the main verb ("Me dothrak" 🡪 "He rides", "Me laz dothrak" 🡪 "He can ride", "Me kis dothrak" 🡪 "He tries to ride"). These have to be regarded as part of the verb phrase but, syntactically, are not preposed to the verb phrase but postposed to the subject of the clause (this makes a difference, e.g., in relative clauses, which have VSO word order instead of the standard SVO word order). The same is true of passive constructions, which are indicated by "nem" postposed to the subject ("Lajak dothrae" 🡪 "The warrior rides", "Hrazef nem dothrae" 🡪 "The horse is ridden"; note that the verb still agrees with the syntactical subject of the clause).

‘VPSlash’ represents a verb phrase missing a complement and kind of sits between ‘V2’ and ‘VP’. It is generated from a ‘V2’ (via ‘SlashV2’ : V2 🡺 SlashVP’) and can then either be turned into a ‘VP’ by adding an object (via ‘ComplSlash : VPSlash 🡺 NP 🡺 VP’) or into a ‘ClSlash’ (clause missing an object) by adding a subject (via ‘SlashVP : VPSlash 🡺 NP 🡺 ClSlash’), to be used e.g. in a relative sentence ("the goat which *the warrior stabbed*"). As such it does not have a complement, like ‘VP’, and retains the ‘objCase’ from ‘V2’, but it does have the ‘subjpost’ field from ‘VP’ (so we can form e.g. "the goat which the warrior *can stab*" 🡪 "dorvi fin *vindee* lajak *laz*").

In order to implement verbal auxiliaries such as "can" and "must", which are represented as ‘VV’s in GF, we also have a dummy lincat for ‘VV’:

VV = {s : Str} ;

Since "can", "must" etc. are simply uninflected particles in Dothraki, this is enough to represent those. This should of course not be considered a final design, since it is unlikely that all verbs with verb phrase complement can be realized like this in Dothraki. However, there is currently very little documentation available on how other ‘VV’ verbs are realized, so this dummy implementation will have to do for now.

### (Relative) Clauses and Sentences

There are no big surprises in the lincats of ‘Cl’ and ‘QCl’ (clauses and question clauses, respectively) or ‘S’ and ‘QS’ (sentences and question sentences):

S, QS = {s : Str} ;

Cl, QCl = {s : Tense => Anteriority => Polarity => Str} ;

The lincats of these categories are fairly canonical in many languages and in Dothraki they are basically the same as in English or German. The main differences between these languages come from whether the linearization also depends on some kind of ‘Order’ parameter, for example distinguishing indirect from direct questions or subordinate from main clauses. It might be necessary to add something like that to our lincat later, if more information about e.g. indirect questions in Dothraki becomes available, but for now there is no indication that we need such an extra parameter.

Relative clauses are more interesting. The relevant categories here are ‘RP’ (relative pronoun), ‘ClSlash’ (a clause missing an object), ‘RCl’ (relative clauses) and ‘RS’ (relative sentence):

RP = {s : QuForm => Case => Str } ;

RCl = {s : Tense => Anteriority => Polarity => Animacy => Number => Str } ;

ClSlash = {s : Tense => Anteriority => Polarity => Str; subj: Str ; objCase : Case} ;

RS = {s : Animacy => Number => Str } ;

Relative pronouns in Dothraki inflect according to the animacy and number of the noun phrase they attach to, and the case that the relativized noun phrase would have been assigned in the embedded relative sentence (similar to German). Hence, the linearization of ‘RS’ depends on the animacy and number of the noun phrase it is eventually attached to and ‘RCl’ is to ‘RS’ what ‘Cl’ is to ‘S’, i.e. it is a relative sentence, whose tense, anteriority and polarity have not yet been fixed. ‘ClSlash’ has to remember the case of the missing object, so it can assign it to the relative pronoun, if used in a relative sentence (via ‘RelSlash : RP 🡺 ClSlash 🡺 RCl’).

There is one more important difference between ‘Cl’ and ‘ClSlash’. When a clause is constructed from a verb phrase and subject via ‘PredVP : NP 🡺 VP 🡺 Cl’, the noun phrase is just prepended to the verb phrase. We cannot do this when constructing a ‘ClSlash’ via ‘SlashVP : NP 🡺 VPSlash 🡺 ClSlash’, because of a quirk in Dothraki word order: Whereas regular main clauses and questions follow SVO word order, relative clauses have VSO word order. So depending on whether a ‘ClSlash’ is eventually turned into a relative sentence (via ‘RelSlash : RP 🡺 ClSlash 🡺 RCl’) or a question (via ‘QuestSlash : IP 🡺 ClSlash 🡺 QCl’), we have to choose the word order accordingly ("dorvi fin vindee lajak" 🡪 "the goat which the warrior stabs" vs "Fin lajak vindee?" 🡪 "What does the warrior stab?"). Therefore, ‘ClSlash’ contains the (linearized) subject in a field, ‘subj’.

### Adjectives

The relevant categories here are ‘A’ (adjectives), ‘AP’ (adjectival phrases) and ‘Comp’ ("complement of a copula", though see below):

A = {s : Degree => Number => ACase => Str ; pred : VForm => Str };

AP = {s : Number => ACase => Str ; pred : VForm => Str } ;

Comp = {s : VForm => Str} ;

There are two basic uses for adjectival phrases: attributive (as in "the strong warrior") and predicative (as in "the warrior is strong"). Attributive use works similarly in Dothraki to e.g. English and German: The adjectival phrase inflects for number and case in agreement with the noun it modifies. In Dothraki the forms for all the cases other than nominative are always identical, however, so we define

ACase = ANom | AOther ;

instead of duplicating these forms in the table.

In English or German, predicative use of an adjective employs a copula ("to be" or "sein"), which is inflected for tense, whereas the adjective is invariant. The same happens in sentences like "the man is a warrior", where the copula "to be" connects the two noun phrases. Dothraki, however, is an entirely copulaless language. When an adjective is used predicatively, the adjective itself is essentially turned into a verb and inflected for tense and polarity ("the strong warrior" 🡪 "lajak haj"; "The warrior is strong" 🡪 "lajak haja"; "the warrior will be strong" 🡪 "lajak vahaja"; "the warrior was not strong" 🡪 "lajak vos ahajo"). Similarly, in a sentence like "the man is a warrior", the ‘NP’ itself inflects for tense (with the ablative and allative form indicating past and future tense, respectively): "The woman is a warrior" 🡪 "Chiori lajak"; "The woman was a warrior" 🡪 "Chiori lajakoon"; "The woman will be a warrior" 🡪 "Chiori lajakaan".

As a result, the category ‘Comp’ (i.e. "strong" in "the warrior is strong" and "a warrior" in "the woman is a warrior") essentially behaves like an intransitive verb in Dothraki. ‘A’ and ‘AP’ have to contain all conjugated forms of the verb as it might be used in a predicative context. Indeed our lincats for ‘A’, ‘AP’ and ‘Comp’, while very much unlike the lincats in the English and German resource grammars, are quite similar to the lincats in the resource grammar for Japanese, another language that does not use a copula for predicative use of adjectives. Here, like in our implementation, ‘Comp’ and ‘AP’ look essentially like verb phrases:

VP = {

verb : Speaker => Animateness => Style => TTense => Polarity => Str ;

a\_stem, i\_stem : Speaker => Animateness => Style => Str ;

te, ba : Speaker => Animateness => Style => Polarity => Str ;

prep : Str ; obj : Style => Str ; prepositive : Style => Str ;

needSubject : Bool} ;

Comp = {

verb : Animateness => Style => TTense => Polarity => Str ;

a\_stem, i\_stem : Animateness => Style => Str ;

te, ba : Animateness => Style => Polarity => Str ;

obj : Style => Str ; prepositive : Style => Str ;

needSubject : Bool} ;

AP = {

pred : Style => TTense => Polarity => Str ;

attr, adv, dropNaEnging, prepositive : Style => Str;

te, ba : Style => Polarity => Str ;

needSubject : Bool} ;

Kommt hier noch was hin?

Figure 3 shows the main categories of the GF Resource Grammar Library and the current status of our implementation. The yellow categories have been fully or at least partially implemented in our resource grammar. For the first milestone, we do not aim to implement every single instance of every category. Instead, we want to first implement the most important categories in a proper way, to make later expansion easier. For example, there are 32 different instances of the ‘mkCl’ function for the clause category ‘Cl’ (Grammatical Framework, 2016b). We did not implement all of them in the current version of our resource grammar, but since there are already some ‘mkCl’ functions included, it is easier to add more functions of the same type later on.

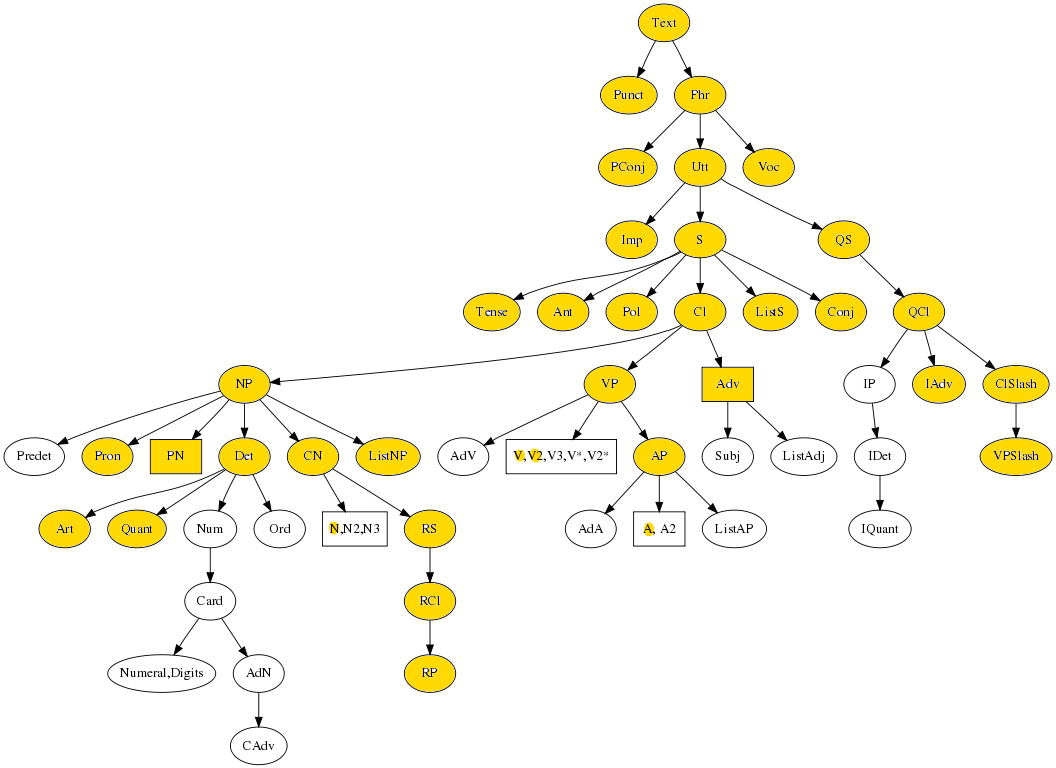


Figure 3: GF Resource Grammar Library Categories (Grammatical Framework, 2016b)

With the current implementation status, it is already possible to parse and translate a wide range of phrases. The following example text is fully supported by the current version of the Dothraki resource grammar:

Beispieltext…

### Operating Instructions

How to use the resource grammar….

## Discussion

### Limitations

Warum nicht alles auf einmal? Zu viel Arbeit, alles gleichzeitig, keine Möglichkeit zu testen, ….

For the implementation of a resource grammar, a good grammar source is needed (Grammatical Framework, 2016b). Although the full resource grammar for the Dothraki language is not yet complete, it is becoming harder and harder to find reliable resources regarding further aspects of grammar we would like to implement (e.g. usage of structural words). As it is a constructed language, and relatively new, we hope that further and more precise information will be documented and published with time.

Yet, even if we had all the information needed for a full implementation, it would not have been possible for us to write a full resource grammar in such a short amount of time, especially as we are still new to the GF programming language and not experienced when it comes to implementing resource grammars. Hence, we focus on providing a good basis of implemented rules, so that we – and others – can expand the project in the future.

Incompleteness/overgeneration of GF approach (grammar rules vs. natural language)….

Regeln versuchen so viel wie mögl. abzudecken, aber nicht immer möglich, siehe Buch? Zitat?...

### Outlook

Expand grammar to a full resource grammar. Community?

### Conclusion

Sinnvolles projekt, ambitioniertes ziel, viel gelernt, nicht einfach, stolz…

## Appendix

### Source Code

The complete source code can be viewed and downloaded via GitHub:

<https://github.com/mahen2/dothraki_gf>

## References

Brown, R., & Frederking, R. (1995). Applying statistical English language modeling to symbolic machine translation. In *Proceedings of the sixth international conference on theoretical and methodological issues in machine translation (TMI-95)* (pp. 221-239).

Cognitus Apps (2016). *Flamingo Dothraki.* Retrieved from https://play.google.com/store/apps/details?id=com.cognitusapps.flamingo.dothraki&hl=de

Fun Translations (2016). *Dothraki translator.* Retrieved from http://funtranslations.com/dothraki

Détrez, G. & Camilleri, J. J. (2016). *Grammatical Framework*. GitHub.com. Retrieved from https://github.com/GrammaticalFramework

Google (2016). *Google Translate*. Retrieved from https://translate.google.com/

Google Groups (2016). *Problem with strange metavariables in parsing*. Retrieved from https://groups.google.com/d/msg/gf-dev/aRjt\_2JfvA0/iTcZbTu5AAAJ)

Grammatical Framework (2016a). Translation with GF: Powered by multilingual grammars. Retrieved from http://www.grammaticalframework.org/demos/translation.html

Grammatical Framework (2016b). GF Resource Grammar Library: Synopsis. Retrieved from http://www.grammaticalframework.org/lib/doc/synopsis.html

Martin, G. R. R. (2016). Bibliography. Retrieved from http://www.georgerrmartin.com/bibliography/

Henkel, M. & Kuckuck, B. (2016). *Building a resource grammar for Dothraki*. GitHub.com. Retrieved from https://github.com/mahen2/dothraki\_gf

Peterson, D. J. (2014). *Living Language Dothraki: A Conversational Language Course Based on the Hit Original HBO Series Game of Thrones*. New York: Living Language.

Peterson, D. J. (2016a). *The Art of Language Invention*. Retrieved from http://www.artoflanguageinvention.com/

Peterson, D. J. (2016b). About Dothraki. Retrieved from http://www.dothraki.com/about-dothraki/

Ranta, A. (2009). The GF Resource Grammar Library. *Linguistic Issues in Language Technology*, 2(2), pp. 1-63.

Ranta, A. (2011a). *Grammatical Framework: Programming with Multilingual Grammars.* Stanford: CSLI Publications.

Ranta, A. (2011b). Grammatical Framework: Programming with Multilingual Grammars. Slides for the GF book. Retrieved from http://www.grammaticalframework.org/gf-book/gf-book-slides.pdf

Ranta, A. (2015). *The Status of the GF Resource Grammar Library.* Grammatical Framework. Retrieved from http://www.grammaticalframework.org/lib/doc/status.html

Tongues of Ice and Fire Wiki (2016). Vocabulary. Retrieved from http://wiki.dothraki.org/Vocabulary

Wikipedia (2016). Grammatical Framework. Retrieved from https://en.wikipedia.org/wiki/Grammatical\_Framework

## Versicherung

Hiermit versichern wir, diese Arbeit selbständig verfasst und keine anderen als

die angegebenen Quellen benutzt zu haben. Wir versichern ferner, dass diese

Arbeit oder Teile davon nicht schon für eine andere Abschlussprüfung oder einen anderen Beteiligungsnachweis eingereicht wurden. Uns ist bewusst, dass ein Zuwiderhandeln gegen diese Versicherung eine Ordnungswidrigkeit darstellt, die mit einer Geldbuße geahndet werden kann.

Düsseldorf, den 16.10.2016

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_