CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Grammatical Framework For Multilingual Natural Language Generation: The Weather Report Case

Master's Thesis in Computer Science: Algorithms, Languages, and Logic

GLEB LOBANOV
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Master's thesis 2017

Grammatical Framework For Multilingual Natural Language Generation: The Weather Report Case

GLEB LOBANOV

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Grammatical Framework
For Multilingual Natural Language Generation:
The Weather Report Case
Gleb Lobanov

Supervisor: Krasimir Angelov, Digital Grammars
Examiner: Aarne Ranta, Department of Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2017

Grammatical Framework For Multilingual Natural Language Generation:
The Weather Report Case

GLEB LOBANOV
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The thesis presents the multilingual natural language generation effort using Grammatical Frameworks and following the Reiter and Dale's approach. Also, it provides an outline for a type-theoretical procedure for document planning, which appears to succumb to formalization as a proof search under a linear context. To demonstrate the applicability of the method to various domains, we created a text robot, which summarizes weather data in English and Russian.

Keywords: Computational linguistics; Multilingual natural language generation; Type theory; Grammatical Framework; Functional programming; Haskell

Acknowledgements

I would like to thank Dr. Krasimir Angelov for his patient and inspirational guidance through my study as well as allowing me to assist in some current research projects. Also, I wish to thank Prof. Aarne Ranta for shaping my understanding of type theory and its applications in computational linguistics. I am very grateful to the Grammatical Framework research group and especially Prasanth Kolachina for valuable advice.

Special thanks go to the Russian government for funding my education at Chalmers.

I would not have been able to complete my master's degree without the ceaseless support of my family.

Gleb Lobanov, Gothenburg, August 2017

Contents

1

Introduction

The ultimate goal of the project is to implement a solution for multilingual natural language generation (MLG) in Haskell and GF, particular parts of which are reusable for MLG enterprises in any domain. The case project is the weather report text robot, which takes weather data from a remote repository and produces consistent reports in English and Russian.

1.1 Motivation Behind MLG

MLG is a way of automatic information representation; others are graphical and tabular which display some data with different kinds of charts and tables. Similarly, texts could be used to facilitate access to compound data. Indeed, texts provide information in a more accessible way because it does not require any additional competence, for example, comprehension of visual descriptions. In other words, it provides a human interface for complex systems which automatically process and summarize a vast amount of information.

Multilinguality promotes diverse access to data sets among people of various mother-tongues. Above all, it is important to provide numerical data or data with complexly organized structure in an easy to understand way. Although English is very popular and influential in the growing global communication, there are still numerous other languages in the world. Thus, more and more people and organizations, both commercial and governmental, discover the need to publish their documents and product information in multiple languages. For these reasons, the development of MLG is a vital and felicitous enterprise.

1.2 Ethical Considerations

Multilingual natural language generation (MLG) is a resource effective technology because it reduces the cost of data summarization and its textual representation: it takes less time and costs less money to produce texts in comparison with approaches involving utilization of human expertise. Besides, an MLG application, which was developed using a functional programming language with dependent types, allows creating proof-carrying documents, which represent information without logical flaws.

On the other hand, MLG carries some threats for society, because it could eventually cause a dismissal of human translators and data analysts. Besides, it could spread misunderstanding across users, because any two words from different languages, translation equivalents, might still represent slightly different concepts in various contexts. Also, not carefully defined logical rules might lead to false documents. It might be inappropriate in some fields, which require transferring a meaning carefully.

MLG possesses a variety of advantages, but its mindless application could lead to some significant problems of information transmission, which a careful study of an application domain could eliminate. Other problems like dismissals of employee and mistrust to text-robots could be solved by requalification of personnel to become supervisors and discourse engineers of MLG systems. All in all, cooperation between humans and assistant text robots is a plausible answer to this arising issues.

1.3 An Outline Of The MLG Architecture

The architecture of the solution is planned along the lines of the Reiter and Dale's framework and includes all three major stages of MLG: document planning, microplanning, and surface realization [34]. Besides, the application rests upon Dannélls' project, aiming to unify, generalize, and extend her implementation of this stages.

Notably, Dannélls' project generates summaries from several already existing databases, but, in a more general case, the input data is series of numbers, which must be analyzed and structured in some way: its artifacts, patterns, and trends must be detected and formalized. Therefore, to establish the first stage—document planning, we took into consideration some ideas from the project, which deals with it effectively—the BabyTalk project [28].

BabyTalk is a natural language generation (NLG) project which attempts to

develop a text robot that summarizes data from the Neonatal Intensive Care Units (NICU). The units output information in several streams of raw data, what is similar to the weather reports case. Then the information from the units goes through three steps of document planning, distinguished by BabyTalks' pipeline: signal analysis, data abstraction, and content determination [28].

We implement this three MLG stages using Grammatical Framework (GF), which is both a functional programming language and a grammar formalism, possesses some valuable features, which make it one of the most helpful tools for MLG [32]. First, it allows programmers and linguists collaborate most effectively. Namely, scholars concentrate on description and development of grammars of several natural languages simultaneously for the same application, and developers use the results through facilities, exhibited by GF as a programming language. Second, the GF is suitable for capturing rhetorical structures, which facilitate text composition. Finally, GF is a logical framework, which allows creating proof-carrying documents, which represent data in logically correct and unambiguous ways. In other words, GF is an appropriate tool for performing the primary tasks of natural language generation, which exhibits multilingual capabilities.

The dual structure of GF grammars, consisting of abstract syntax and multiple concrete syntax, allows building MLG application according to the Reiter and Dales's approach. The abstract syntax describes the lexicon of an application domain, captures rhetoric structures, and encodes the system of shared linguistic concepts common to several languages. Concrete syntax encode linguistic information about certain natural languages, using one shared abstract syntax. The procedure of linearization, defined in GF, produces texts for given sentences, which are encoded in concrete syntax. Accordingly, abstract syntax is convenient for the formalization of the document plan and the document structure.

1.4 Results

To demonstrate capabilities of Grammatical Framework for multilingual natural language generation, we created a demo application, which extracts data from a web database and generates textual weather reports. It summarizes data sets containing information about the state of the environment during a specified period in English and Russian.

Usually, weather information is provided either in tables or short sentences, written by meteorologists, but there are also some benefits from the use of text

robots. Automatization of the weather report summarization provides direct distribution of information in an accessible form. Besides, it cuts down expenses since it does not involve human expertise and supervision. The addition of multilingualism could increase its user coverage and make it, even more, resource efficient, allowing to leave out the expensive professional translation from the process.

To produce a report, a text robot must know its structure, a document plan, in advance. It contains data as well as information about how to group it and can be defined beforehand or generated automatically. It is a kind of template, which carries semantics and logic of the text.

During the project, we created a mechanism for automatic production of document plans. Before, GF MLG applications needed predefined templates of the whole text. Now, to set the final communicative goal of the text generation, a programmer needs only to describe semantics and short templates for sentences, building blocks of the text. A document plan of the text, then, is constructed automatically with regards to semantic and logical rules of the domain.

To accomplish this goal, we modified the standard proof-search algorithm of GF. Being essentially an implementation of Martin-Löf type theory, GF allows proving theorems. It is a functional programming language with dependent types, type checking mechanism of which is used for this purpose. The communicative goal is a theorem, and its document plan is its proof, so these facilities would do the job of automatic document planning after some extensions.

The results of this experimentation were presented at the *Workshop on Logic and Algorithms in Computational Linguistics 2017* [22], and the shorter and partly changed version of this thesis was published in its proceedings.

2

Frameworks

2.1 Grammatical Framework

Grammatical Framework is a functional programming language with dependent types and a grammar formalism, which is being developed with the purpose of defining natural language grammars in mind [32]. It is used in machine translation and natural language generation applications. Aarne Ranta initiated its development in his fundamental work *Type-Theoretical Grammar* [29], where he used it as a notation to express the semantics of natural languages using Martin-Löf type theory. Its first implementation was reported in [27], and, since then, it has been evolving as a functional programming language. Now it is a fully fledged tool, which is utilized for research on computational linguistics and application development.

The key feature of GF is the separation of abstract syntax from concrete syntax, which makes the development of multilingual applications more efficient. Abstract syntax describes semantics or the structure of a text, and concrete syntax describes the grammar of its languages. It allows division of labor among programmers and grammarians, who can now pay special attention to their areas of responsibility. Besides, GF can easily connect one abstract syntax to several concrete syntax, which makes multilingualism accessible and effortless to scale.

GF focuses on linearization, which renders functional rules of abstract syntax into strings as specified by rules of concrete syntax. This fact makes it a useful tool for natural language generation tasks.

2.1.1 Abstract syntax

The GF facility for abstract syntax description is the implementation of Martin-Löf's intuitionistic type theory, which is essentially the lambda calculus with dependent types or Logical Framework. For this reason, it allows not only to type check but also design proof-carrying documents. This capacity can, for instance, be used

for natural language generation tasks associated with the essential requirement to produce logically correct texts. Moreover, the way abstract syntax is connected to concrete syntax preserves the meaning among multiple languages.

Abstract syntax consists of type declarations for functions and categories, which represent semantic entities and relations among them. Below, the classical food grammar example illustrates the design of a regular abstract syntax, which is taken from the Grammatical Framework Tutorial [31].

```
abstract Food = {
      flags startcat = Phrase ;

      cat
        Phrase ; Item ; Kind ; Quality ;

      fun
        Is     : Item -> Quality -> Phrase ;
        This   : Kind -> Item ;
        These  : Kind -> Item ;
        Apple  : Kind ;
        Apples : Kind ;
        Very   : Quality -> Quality ;
        Ripe   : Quality ;
    }
```

Here four categories, the basic types, are declared: `Phrase`, `Item`, `Kind`, `Quality`. The type `Phrase` is the start category, which means that it will be used as the default category for generation and parsing. These categories are used as the argument or target types of the abstract syntax functions `Is`, `Very`, and so on. An example of expression of type `Phrase` is the function application `Is (This (Apple)) (Very Ripe)`.

2.1.2 Concrete syntax

Concrete syntaxes are described with a purpose-built GF tool, which is lambda calculus with records and is, also, kitted up with string processing functions. Also, the GF Resource Grammars Library API can be used for assistance. It includes basic lexis, morphology, and syntax of 32 languages [30]. Most important, the concrete syntax functions firmly correspond to the abstract syntax functions of a

particular application and even keep their names and types from one concrete syntax to another, so uniformity of grammars for all defined languages are retained for this application.

Below is the concrete syntax for the food grammar, which determines how the expression of its abstract syntax is linearized to a string or parsed. Thus, the expression above will correspond to a line *This apple is very ripe*.

```
concrete FoodEng of Food = {
     lincat
       Phrase, Item, Kind, Quality = {s : Str} ;

     lin
       Is item quality = {s = item.s ++ "is" ++ quality.s} ;
       This kind = {s = "this" ++ kind.s} ;
       These kind = {s = "these" ++ kind.s} ;
       Apple = {s = "apple"} ;
       Apples = {s = "apples"} ;
       Very quality = {s = "very" ++ quality.s} ;
       Ripe = {s = "ripe"} ;
}
```

As can be seen from this example, the primary object of concrete syntax is not a string but a record, which contains a string. A record can contain more than one element, and this element can also be a table. Tables map grammatical parameters into strings, other records, or tables. These capabilities allow detailed customization of the linearization of abstract syntax functions.

In some cases, this grammar produces incorrect strings. Since both `Apple` and `Apples` are of type `Kind`, the expression `Is (This (Apple)) (Very Ripe)` is well-typed and linearized to the grammatically incorrect sentence *This apples is very ripe*. Below is the extended and modified concrete syntax of the food grammar, which illustrates how to use tables and records to solve this problem.

First, we determine a parameter `Number`. Then we define the linearization of the category `Kind` to be a table, which maps the `Number` parameter to strings. Also, to the record `Item`, we add the field `n` of type `Number`. Finally, we give new definitions for the functions `Apple` and `Is`; we remove the separate function for the plural form `Apples`. Using the bang operator, we get access to the values of tables, like in the linearization of `These`.

```
param Number = Sg | Pl ;
lincat Kind = {s : Number => Str} ;
lincat Item = {s : Str ; n : Number} ;
lin Apple = {
      s = table {
        Sg => "apple" ;
        Pl => "apples"
      }
} ;
lin This kind = { s = "these" ++ kind.s ! Pl ; n = Sg } ;
lin These kind = { s = "these" ++ kind.s ! Pl ; n = Pl } ;
lin Is item quality =
    {s = item.s ++ table {
        Sg => "is" ;
        Pl => "are"
    } ! item.n ++ quality.s
} ;
```

2.1.3 Abstract Syntax For Document Planning

Also, we use the abstract syntax to model the discursive structure of the text, which is called a document plan. Following the notation of Rhetorical Structure Theory [23], it consists of text spans which are either nuclei or satellites combined in a variety of relations. For example, the most common used relation in our project is `Background`, which gives an account of a process, a state, or an action in the main, nuclear, message. Functions, arguments of which are constituent messages, represent such relations.

The messages, in turn, have the complex structure, which is too captured by the abstract syntax. They could be either atomic or composite. The first denote entities or abstract concepts; they are primitive building blocks. Every field of a data object, received by the application corresponds to such message. From the concrete syntax perspective, this messages catalog the lexicon of the weather domain. Functions with only one argument, which is the numerical or string value of the corresponding field, represent these messages.

Composite messages combine atomic messages and, after linearization to natural languages, are text spans, nuclei or satellites, which are bind together and form rhetorical relations. More detailed exposition of abstract syntax organization is in section *4.1 Abstract Syntax*.

The standard GF proof search algorithm, after some extensions, automatizes document planning. Provided only a communicative goal and a list of atomic messages, it constructs a plan of a text, which is ready for linearization. This problem is central to the current project, and its solution is presented in *Chapter 3*.

2.2 The MLG Framework

Our primary source for multilingual natural language generation (MLG) is *Building Natural Language Generation Systems* by Reiter and Dale [34]. Although it was published in 2000, it remains the only work bringing together all the key aspects of the topic. Moreover, the authors present a comprehensive theoretical framework for future natural language generation (NLG) projects. Later, Bateman and Zock describe it in the reputable Oxford Handbook of Computational Linguists in the chapter "Natural Language Generation" [9], and, also, Dannélls put it in practice for the project that generates artwork descriptions [13], what demonstrates significance and functionality of the approach.

Reiter and Dale distinguish three stages of NLG. The first stage, document planning, determines the content and the structure of a document. The second stage, microplanning, takes the document plan and constructs a document specification of syntactical structures and lexicon of the document. The third stage, surface realization, produces actual text using the document specification. Between stages, trees, which store structural information in internal nodes and content information in leaves, transmit the document plan and the syntactical specification.

2.2.1 The Weather Report MLG Pipline

The Weather Report MLG pipeline is an implementation of Reiter and Dale's [34] architecture, which Portet and his colleagues put into practice for generation of textual summaries from neonatal intensive care data [28]. They analyze health data: filter artifacts, recognize its patterns and trends. Then, following the standard strategy, they abstract the data, determine the content of the final message. Then, after microplanning, this message is realized. Taking into account that GF covers

the last two steps, we distinguish the following procedures in our system.

First, the system receives the data to analyze. We are going to use meteorological information in the form of some time series containing information about soil and air temperature, humidity, wind direction and so on; the period taking into consideration can vary. During the signal analysis stage, we detect artifacts, patterns, and trends. Then this information is encoded regarding the meteorological domain ontology, which is represented in the system as a Haskell algebraic data type.

Second, the captured and formalized data must be interpreted to produce general observations of the state of affairs. This step involves reasoning based on both ontological properties and explicitly defined rules. The properties and rules are defined in Haskell, and the result of the interpretation is a refined local ontology encoded using a respective Haskell functions and algebraic data types.

Third, the local ontology is transformed into the tree, the document plan, of linked events and states. The processing is conducted according to the rules that specify, what information is most valuable and in what sequence it must present. Both the rules and the resulting tree are encoded using Haskell functions and algebraic data types.

Fourth, microplanning step transforms the events and states tree to the tree that represents the syntactical structure of the documents, also carrying its lexicon. The tree is encoded in GF abstract syntax in two passages: the transformation of a document plan to the system of rhetorical structures and lexical items, and then the transformation of the previous result to the system of the syntactical structures using the GF resource grammar library [30].

Finally, the actual texts in several languages are generated utilizing GF linearization using GF concrete syntax. The GF infrastructure entirely covers this step. Indeed, our responsibility on this step is to provide concrete syntax of the languages we want the system to generate the reports—English and Russian. Since the lexical aspects of the meteorological domain have not been explored in GF research yet, concrete syntax and lexicons of both languages were extended. As well, some auxiliary syntactic functions were added for both languages to carry out some markup actions. Also, GF Resource Grammar does not provide for the control of the free word order in Russian, so some additional functions, which handle it, were written.

2.3 Related MLG Applications and Tools

Here related MLG applications are listed. Some of them have MLG as only a necessary phase, and the others have MLG as their primary purpose. Not all of them use GF. More applications, which use GF, can be found on the dedicated page of the GF website [12].

2.3.1 The Météo system

The Météo system was a machine translation system for the weather forecasts, which was being developed by the University of Montreal's Automatic Translation Research Team from 1975 to 1977. It was incorporated into the Canadian forecasts transmission network and translated from English to French. It processed several types of forecasts from the eight regional meteorological offices. From then on, Environment Canada used it successfully until 2001, when it was replaced by another application [11] [36].

The system had some distinctive features. Its application domain model of natural language for weather forecasts had restricted vocabulary and syntax, and its dictionary and grammar were separated from the algorithm component of the software. When the automatic translation was impossible due to the absence of a word in the dictionary or ambiguity of syntactic structure, the system submitted a phrase under consideration for human translation. Nevertheless, if it was possible, the system tried to provide for its partial translations [35].

For its time, Météo was the unique and advantageous application. It provided reliable translation because professional translators were involved in its developments. They provided invaluable feedback during all phases of the application construction: design, development, and refinement [35]. Besides, after more than twenty years of continuous operation, its yearly throughput reached to 30 million words during the final years [11]. Its success makes this system the primary reference point for any further MLG weather report applications.

2.3.2 MLG for the Gothenburg City Museum

Dana Dannélls as part of her Ph.D. project created a text robot which generates short descriptions of artworks in English, Swedish, and Hebrew for Gothenburg City Museum [13]. It takes as input data base records containing information about

paintings: the artist, the name, its type, what does it depict, and so on. The output is several sentences expressing this information in a nice human-readable manner.

GF is the core technology of Dannélls' project. It follows Reiter and Dale's [34] NLG framework and uses static templates for the document planning phase. The weather report application has a similar design; nevertheless, we have developed a new approach to document planning, which is presented in *Chapter 3*.

2.3.3 Generation of a natural language formal proof of the correctness of insertion sort

Thomas Hallgren provided a proof of the correctness of insertion sort, which was constructed formally in the Proof Editor Alfa [4] and, then, translated automatically to English [19]. With this experiment, Hallgren inquiries into the advantages and disadvantages of automatically generated natural language proofs. Its positive aspects are multilingualism, which could be easily achieved, consistent terminology, and the close correspondence to the formal proof. However, the text could become tedious to comprehend and abnormal expressions.

2.3.4 Translation of formal software specifications to natural languages

As a part of the KeY system [2], Kristofer Johannison developed a tool, which translates formal software specifications in OCL [3] to natural languages[20]. This tool is supplemented by a syntax-directed editor, which enables to develop specifications in OCL and natural language simultaneously. The main motivation behind this project is to make development and maintenance of software specification unchallenging to the layman. GF is the core technology of the system.

2.3.5 WebALT: Multilingual Tests in Mathematics

The WebALT [10] is a system for generation of automatic interactive assessment applications for mathematics education. Several types of questions are supported: multiple choice, yes/no questions, and custom solutions, which are checked automatically. Using a special language independent encoding, one can compose an assignment, which is, then, translated into several natural languages in written and verbal forms: English, Spanish, French, Italian, Swedish and Finnish. Also, there

exists the WebALT Excercise Repository, which makes it possible to construct an international syllabus using some predefined learning objects.

2.3.6 GF Offline Translator

Krasimir Angelov, Björn Bringert, and Aarne Ranta developed a speech-enabled hybrid multilingual translation application for mobile devices [8][16] *GF Offline Translator*, which is available on *Play Store* [17] and has a web version [15]. It is a rule based alternative to statistical machine translation of Google Translate. It supports 14 languages and 182 language pairs: Bulgarian, Catalan, Chinese, Dutch, English, Finnish, French, German, Hindi, Italian, Japanese, Spanish, Swedish, Thai. Both textual and audio input is available.

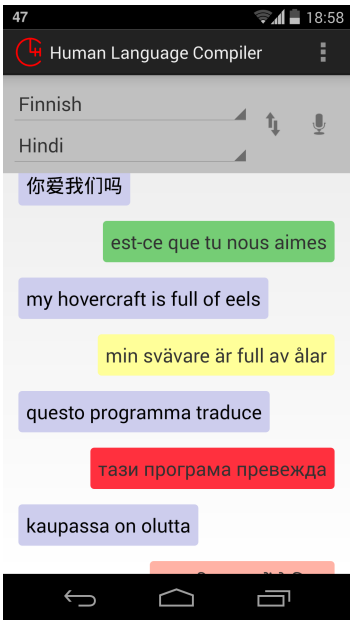

Figure 2.1: A screenshot of a mobile application *GF Offline Translator* [16].

Its distinctive feature is the ability to indicate confidence of translation with colors as shown in figure 2.1. Green color indicates semantically correct translation. Yellow color indicates grammatically correct translation, but it does not guarantee the precision of meaning. Red color indicates that the input string was split into chunks or even words, which is indicated by darker shade. It means that these pieces were translated one by one and separately. In all cases, answers could be accompanied with alternative translations. Another feature is the ability to work offline.

2.3.7 The online editor for simple multilingual grammars

The online editor was created for simple multilingual grammars. It makes it easier to construct GF grammars for people without experience in programming [5]. It has a straightforward interface, so a beginner could embark on grammar construction without going through sometimes cumbersome software installation and setup. After the completion of grammar, it can be tested in the web application, which allows constructing sentences from the given lexicon and grammar, or with an automatically generated quiz.

3

Document Planning

Document planning is the essential element of the MLG pipeline, which given the communicative goal determines the content and the structure of an output document. Other phases, which influence the shape of a document, are microplanning and linearization (surface realization). Possible flaws of this latter phases do not cause fatal errors. At the same time, users expect to receive a coherent text covering all essential details responding to a provided query. Thus, in our approach, we see document planning as the central task. It is a search problem, which yields an abstract expression, constituents of which represent complete and requisite information. To solve it we revised the common GF proof search algorithm [6].

3.1 The Standard GF Search

The standard GF search consumes a target type and returns either a random subset or all abstract expressions of that type. They can be used to check if their linearizations are semantically and grammatically correct. Additional regulation could be imposed on the choice of abstract expressions. A statistical model is used to set selection probability for an expression [7]. Regarding the exhaustive production of all possible expressions, it gives the result in the decreasing probability order. Also, the choice could be controlled by constraints encoded using dependent types since GF uses a Logical Framework, a version of constructive (also known as intuitionistic) type theory [24], for a description of the abstract syntax.

The search algorithm in GF employees the choice sequence interpretation of type theoretical terms [25]. It takes a meta variable expression of the initial type. Then, on every step the left most meta variable is substituted with a function application expression, the target type of which must be unified with the type of this meta variable. At the same time, arguments of this function application, new meta variables, must saturate its type. On the first step, the left most meta variable is the

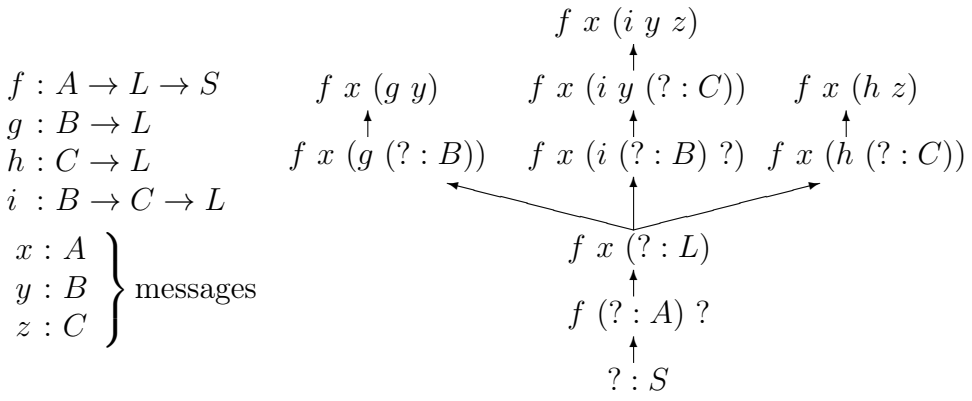

Figure 3.1: An example of abstract syntax and its corresponding choice tree

initial one. Since for some meta variables, there could be choices, the algorithm constructs a choice tree, which represents all possible expressions, possible states. The final states are complete expressions, which don't have constituent meta variables.

Figure 3.1 illustrates a choice tree for abstract syntax without dependent types, all functions of which have only first order types. Functions can also have as well high-order types and take as arguments other functions, lambda abstractions. In a linguistic domain, they are used for modeling anaphoric references [32]. As for dependent types, they manifest complex semantic constraints. Since the presence of this features does not affect the design of the planner, which also relies on their existing implementation in Grammatical Framework, they are excluded from the example.

In the example, first, we start with a meta variable of the type S, which is the target type of expressions we are searching for. It is replaced with the application $f\ ?\ ?$ because it is the only choice given in the grammar on the left hand of the figure. Then, we continue picking the left most meta variable, annotated with its type for convenience, until a complete term is reached. The resulting term is a leaf of the choice tree.

The third step produces three alternatives for a function of type L and creates three branches in the tree, leading to three different complete expressions. If our goal is to generate a random expression, the existing GF search algorithm chooses a branch randomly. Alternatively, for the exhaustive search, the algorithm uses breadth-first priority traversal.

3.2 The Document Planning Search

In document planning, we search for subtrees consisting of a given list of messages, representing information that we want to communicate. These messages correspond to basic types of the abstract syntax: A, B, and C; values x, y, and z of this types carry their content (see 3.1). To make example simple, we provide here only one value for each type, but it is not mandatory. The initial type S, from which we start to construct a document is the communicative goal, which to some extent sets a shape of the resulting expressions. To generate texts for different goals, we need different initials types.

The query is expressed as a linear logic multiplicative conjunction, operands of which are either pair of an expression and its message type (e.g., $x : A$) or negations of a message type (e.g., $\neg A$). The values specified in pairs are always used, while the use of negated types is forbidden. Types that are not included in the query are saturated with arbitrary functions which have them as their target types. Their repetition in the resulting expressions is not restricted.

To fulfill a query is to find all complete expressions which consist of all expressions provided in a conjunction and at the same time do not include any expressions of negated message types. Since Grammatical Framework does not distinguish between ordinary basic types and message basic types, a query determines which of them are considered as messages, but the framework processes them without priority.

Execution of the query $(x : A) \otimes (y : B)$ treats types A and B as messages. Values of these types are used in expressions $f\ x\ (g\ y)$ and $f\ x\ (i\ y\ z)$. The type C is not included in the query conjunction as a negated message type, so in the second expression we are free to use $z : C$. On the other hand, if we would like to treat C as a message type, we can either provide a pair $z : C$ or a negation $\neg C$ to a query. In the latter case a query is $(x : A) \otimes (y : B) \otimes \neg C$, and the only complete expression satisfying it is $f\ x\ (g\ y)$.

Aggregation of messages is conditional on the grammar and the query. To place messages x, y, and z into the resulting expression simultaneously, we use the following query:

$$(x : A) \otimes (y : B) \otimes (z : C)$$

It returns the expression $f\ x\ (i\ y\ z)$ where the function i uses y and z to represent a completed sentence, which could communicate information embedded in both messages.

3.3 Implementation

The extension of the GF search algorithm for document planning involves the addition of a mechanism that carries a query during the construction of the choice tree. The query is used when the type of the meta variable under consideration could be found in the query, and it must be paired with an expression in the query conjunction. Then, instead of employing a free choice, this suitable expression replaces the meta variable. In the final expression, every message must occur only once; therefore, the related pair in the conjunction is replaced with the negation of its type.

If the type of the current meta variable is negated in the query conjunction, the current branch is abandoned, and traversal of an alternative branch begins. On the other hand, if this type is not in the query at all, a value of the current meta variable is chosen freely from alternatives provided by the structure of the choice tree.

The search halts and returns the resulting complete expression if and only if all types of the query conjunction are negated. If it halts with a complete expression, but some pairs of expressions and types remain in the query, this expression is not accepted.

Below is an outline of generation of $f\ x\ (i\ y\ z)$ followed by the corresponding query after every step:

$? : S$	$(x : A) \otimes (y : B) \otimes (z : C)$
$f\ (? : A)\ ?$	$(x : A) \otimes (y : B) \otimes (z : C)$
$f\ x\ (? : L)$	$\neg A \otimes (y : B) \otimes (z : C)$
$f\ x\ (i\ (? : B)\ ?)$	$\neg A \otimes (y : B) \otimes (z : C)$
$f\ x\ (i\ y\ (? : C))$	$\neg A \otimes \neg B \otimes (z : C)$
$f\ x\ (i\ y\ z)$	$\neg A \otimes \neg B \otimes \neg C$

For one query, the search could output multiple complete expressions satisfying restrictions imposed by grammar and a conjunction of messages and message types. In this case, the grammar must be designed in a particular way to avoid

ambiguity. All expressions must communicate the same information for the same message arguments, but their linearization could vary. It could also depend on additional arguments—syntactic categories. This feature adds variability for a planner if it randomly chooses the resulting expression from the multiple search output.

4

The Weather Report Case

As an illustration for document planning, we have chosen the weather report case, which is a practical application involving uncomplicated analysis of the data at hand. It exhibits grammar-construction principles without distractions caused by peculiarities of more different domains with hard-to-comprehend specific languages. The application receives data in a JSON format from DarkSky and builds a document plan, which is according to [34] a tree, consisting of information-bearing messages and discourse relations between them.

In GF a document plan is a tree built from applications of functions in the abstract syntax of the weather report grammar. The concrete syntax define rules for linearization (surface realization) of a document plan, a process resulting in the final text, a weather summary, in English and Russian. They use GF Resouce Grammar API [30] calls, a library which covers morphology and basic syntax of 32 languages, and the application domain lexicons.

4.1 Abstract Syntax

The abstract syntax of the weather report grammar is modular, reflecting the compositional structure of the text. It comprises three modules: atomic messages, composite messages, and rhetorical structures. Atomic messages are functions with one or no arguments, which stand for one value or an abstract concept from the raw data source. The type of the value is the target type of the corresponding function. Composite messages represent phrases or sentences in the text and are functions with several arguments which are applied to either atomic messages or other composite messages. The target types of the composite messages depend on their roles in the rhetorical structures. The latter is defined in the third module, which consists of functions bringing together text spans and producing the schemata of the Rhetorical Structure Theory [23].

4.1.1 Atomic messages

As we said atomic messages stand for a value from the raw data source. They could also represent abstract concepts, qualities or other bits of information which aren't supported by values but are necessary constituents of discourse. For example, the following are two of the atomic functions which correspond to values from Table 4.2, which shows the main data values used in the weather report production:

```
TemperatureVal : Float -> Temperature
ExtremelyHot : TempType
```

Here, function `TemperatureVal` takes a real number and returns a value carrying message of type `Temperature`. In contrast, function `ExtremelyHot` does not require any arguments and has target type `TempType` denoting human perception of temperature levels.

4.1.2 Composite messages

When linearised, composite messages are phrases or sentences. They consist of atomic messages, and they use them to deliver a complete utterance about some state of affairs, action or process. In the abstract syntax, they are formalized as functions with one or more arguments, where the types of the arguments must be target types of functions for atomic messages. Target types of composite messages are either `Nucleus` or `Satelite` – concepts from Rhetorical Structure Theory, as will be discussed later. An example of a composite message is `InfoPrecipType`:

```
InfoPrecipType : PrecipIntensity -> PrecipType -> Satellite
```

It takes an atomic message `PrecipIntensity`, which is a short description of precipitation intensity together with its value (millimeters of water), and an atomic message `PrecipType`, which stands for a precipitation type, e.g., snow, rain, and so on. The target type `Satellite` signifies the role of the composite message which it plays in more complex rhetorical structures defined in the abstract syntax.

4.1.3 Rhetorical Structures

The next structural level after the messages is the rhetorical structure. Rhetorical Structure Theory (RST) [23] describes text structure regarding relations between its constituents. Each relation holds between a core element, a nucleus, and a set of supporting elements, satellites. A theory exhibits a comprehensive system of schemata describing different combinations of relations, nuclei, satellites, and

Circumstance	Solutionhood
Elaboration	Background
Enablement	Motivation
Evidence	Justify
Volitional Cause	Non-Volitional Cause
Volitional Result	Non-Volitional Result
Purpose	Antithesis
Concession	Condition
Otherwise	Interpretation
Evaluation	Restatement
Summary	Sequence
Contrast	

Table 4.1: The list of RST relations [23].

constraints on them. Table 4.1 enumerates the full list of relations as presented in [23].

Below, there is an example of a schema formalization, which has `Background` as a core relation:

```
cat Nucleus
    Satellite
    SatelliteList
    Schema


fun
  BSat : Satellite -> SatelliteList
  CSat : Satellite -> SatelliteList -> SatelliteList

  Background : Nucleus -> SatelliteList -> Schema
```

We have settled on RST because it does not depend on any particular language and text type or purpose. This feature is beneficial because the weather report case produces weather summaries in English and Russian. Besides, it has a wide recognition in an NLG community [18]. Alternatively, [26] outlines and formalizes a document structure using a type theory with dependent types. His results combined with RST could be appropriate and practical for our approach, taking into account that GF is essentially a logical framework — a lambda calculus with dependent types. However, for the weather report case, we confined ourselves with some basic RST concepts.

4.1.4 Dependency groups

The proof-search algorithm employed for the document planning has a property that it selects a composite message to use in text generation only if all of its constituents are present in a generation environment or context, which is a list of available and yet unused atomic messages. For instance, if `PrecipIntensity` is absent in an environment, `InfoPrecipType` will not be chosen. Then, if `PrecipType` is present in an environment, it is left to be used by other composite functions. Besides, if no composite messages which use it are found, the algorithm returns nothing. In other words, to produce text, all atomic messages must meet whichever composite message functions which take them as arguments.

This behavior of the algorithm leads us to group atomic messages according to, first, our wish to see them in one composite message simultaneously, and, second, interdependency of their appearance in an environment. The former assertion is grounded in mechanics of the algorithm and is evident from the earlier example. The latter one is because some atomic messages are computed from others. Thus, in some cases, they could either be excluded from data by a data provider or just be meaningless and, therefore, undefinable. For instance, a type of precipitation `PrecipeType` depends on a value embedded in `PrecipIntensity` in a sense that if the intensity is equal to zero, a type of precipitation is unidentifiable. Moreover, in our case, the data provider marks it as an optional field, and we could have a situation in which `PrecipeType` is undefined while `PrecipIntensity` is present. A workaround is to join them to a dependency group and allow them to appear in the list of available messages only when both of them are present.

4.1.5 Requirements of grammar completeness

For a grammar to be complete, it must comply with the main rule which guarantees its robustness: each dependency group must be represented by at least one composite function which has all elements of this group as arguments. At the same time, all arguments of a composite function must be members of the same dependency group. Regarding independent atomic messages, they are regarded as singleton dependency groups and, thus, must be supplied by a composite function of one argument each.

If there is a need to make use of several dependency groups in a composite message simultaneously, they should be combined into one group, which could be interpreted as a disjoint union. Since it is possible to ignore atomic messages during linearization, it will not restrict granularity. Instead, it makes possible to produce

texts, which communicate only subsets of dependency groups. Another feature, increasing flexibility, is that every dependency group can be represented by more than one composite message.

4.2 Implementation

The implementation consists of three parts: a GF grammar, the new document planning algorithm in the GF runtime, and a host Haskell application, which receives weather data, constructs, calls the planner and produces output text. Below are example fragments in English and Russian followed by the corresponding abstract syntax tree.

```
On Sunday, 16 April 2017 at 16:08 in Gothenburg it is snowing.
It is very cold: the temperature is 4.22 °C, and it feels
like 0.95 °C.

В воскресение, 16 апреля 2017 16:08 в Гётеборге снежно.
Очень холодно: температура 4.22 °C и 0.95 °C по ощущениям.

Background (InfoLocation Gothenburg
                         (DayVal "16") April (YearVal "2017")
                         Saturday (TimeVal "16:09")
                         IconClearDay)
           (BSat (InfoTemperature VeryCold
                                  (TemperatureVal 4.22)
                                  (ApparentTemperatureVal 0.95)))
```

The grammar is compiled to a PGF file, which contains a portable grammar, accompanied with a Haskell interface. This file is automatically generated source code. Together with the PGF Haskell library [6], the interface facilitates abstract syntax tree manipulations. The library also contains the document planning algorithm, a modified proof search algorithm.

The host application receives, verifies, and processes raw data from [14], a weather forecasting cloud service. A user inputs the name of a location, weather conditions of which are to be displayed. The application queries Google Maps API [1] for its coordinates. Then, these coordinates are used to get a JSON object with weather information from DarkSky. Future, current, and historical data are stored there, which accessible by accompanying the coordinates with a particular date.

If some fields of the JSON object are absent, we mark that by using a negative term in the query. The presence of a negative term in a dependency group will not allow using the whole group in the discourse. Granular partitioning of dependency group makes the application sustainable in case of some failures in the weather data.

Some values undergo processing; for example, temperature or wind speed, are translated from imperial units to metric. Some JSON fields that we use in the current implementation are listed in Table 4.2. Then, values are transformed to atomic messages and supplied to the document planning algorithm, which returns an abstract syntax representation of a weather report ready to be linearized to English or Russian using the standard PGF API.

Since there are no predefined templates, which fully cover the text structure, the composition of reports varies on every run of the application. The first sentence, which contains information about a location, remains the same, but the order of other sentences defers. During one run, the compositions of English and Russian texts are identical.

The source code of the application is published on GitHub [21].

Field	Description
temperature	The air temperature in degrees Fahrenheit
cloudCover	The percentage of sky occluded by clouds, between 0 and 1, inclusive.
dewPoint	The dew point in degrees Fahrenheit.
humidity	The relative humidity, between 0 and 1, inclusive.
icon	A machine-readable text summary of this data point, suitable for selecting an icon for display.
ozone	The columnar density of total atmospheric ozone at the given time in Dobson units.
precipIntensity	The intensity (in inches of liquid water per hour) of precipitation occurring at the given time.
precipProbability	The probability of precipitation occurring, between 0 and 1, inclusive.
precipType	The type of precipitation occurring at the given time.
pressure	The sea-level air pressure in millibars.
time	The UNIX time at which this data point begins.
windBearing	The direction that the wind is coming from in degrees, with true north at 0° and progressing clockwise.
windSpeed	The wind speed in miles per hour.

Table 4.2: Some fields of a JSON-response object provided by the API of [14]. We list those that are used in the current implementation of a text robot. Some of them are optional or belong to a data-point object, and the application handles their faulty absence. All listed fields are transformed to related atomic messages after the data is received, verified, and processed. Descriptions are taken from The DarkSky's API documentation.

5

Conclusion

This work has demonstrated that a small-scale modification of the standard GF proof search algorithm could facilitate the document planning phase of the natural language generation pipeline. Its automatization saves from labor intensive description of conditional rules for template population, which could become exhausting when they are comprehensive and numerous. For the same reason, it also makes available variability of the output text at no cost because several proof objects could be found for the same target type.

By using randomized search and some redundancy in the abstract syntax of the grammar, we could also achieve variability in the generated text. Not only wording but also the order of sentences and rhetorical relations can vary. The variability could be beneficial in the case when texts, which convey similar information, are generated repetitively and in large quantities. Without monotonous wording, the overall user experience becomes more satisfactory.

During the weather application grammar construction, its general and feasible architecture together with the requirements for its completeness have been outlined and formulated. We classified all messages to atomic and composite and grouped the atomic messages according to their appearance in the data source. Consequently, the obligation was imposed on the composite messages to contain atomic messages from only the same dependency group. This regulation organized the data and secured coherent communication of information.

The framework is multilingual, so it also allows the same report to be rendered in several languages at once with the preserving of the meaning across them. We showed it by the example of parallel linearization of the same information to Russian and English. Moreover, being a rule-based system, it gives predictable output. In comparison with systems, which use statistical methods, such as Google Translate, it guarantees a publishing quality and targets producers rather than consumers [33]. The weather report production is an example of such task.

The weather report case illustrates practical details of the approach and is an evidence of its straightforward implementation. It will be published online and serve as a demonstration of how GF could be employed for MLG tasks.

5.1 Further Work

Our next objective is to transform the apparatus that underlies the weather report application to a Haskell general library for MLG. It will provide programmers with GF utilities and resources through API, which does not require them to be acquainted with GF internals. Its architecture will reflect common assumptions on how texts should be structured discursively, semantically, and syntactically. As a result, programmers could transparently bring expected functionality to the end users of the application, developed using this library.

Another task is to upgrade the document planning algorithm. Now, to get a document plan, we must supply to the algorithm a communicative goal together with a list of atomic messages, which carry information to be conveyed. A different approach, which we believe is possible to carry out, is to supply only the list of atomic messages, and the refined algorithm will figure out what function will express the communicative goal. It will use types of atomic messages and try to find the appropriate function from a prespecified inventory. Then, it will apply it to this atomic messages and random or default values, and, finally, constructs the document plan.

Bibliography

[1] Google maps APIs | Google Developers. `https://developers.google.com/maps/`. (Accessed on 07/31/2017).

[2] The key project. `https://www.key-project.org/`. (Accessed on 08/23/2017).

[3] OCL Portal. `http://www-st.inf.tu-dresden.de/ocl/`. (Accessed on 08/23/2017).

[4] The Proof Editor Alfa. `http://www.cse.chalmers.se/~hallgren/Alfa/`. (Accessed on 08/23/2017).

[5] John Camilleri Ramona Enache Aarne Ranta, Thomas Hallgren. D2.2 grammar ide. *MOLTO Deliverable*, 2011.

[6] Krasimir Angelov. *The Mechanics of the Grammatical Framework*. PhD thesis, Chalmers University of Technology, 2011.

[7] Krasimir Angelov. *Probability Distributions in Type Theory with Applications in Natural Language Syntax*, pages 279–296. Springer International Publishing, 2017.

[8] Krasimir Angelov, Aarne Ranta, and Björn Bringert. Speech-enabled hybrid multilingual translation for mobile devices. In *European Chapter of the Association for Computational Linguistics*, Gothenburg, 2014.

[9] John Bateman and Michael Zock. Natural language generation. In *The Oxford handbook of computational linguistics*. 2003.

[10] Olga Caprotti and Mika Seppälä. Multilingual delivery of online tests in mathematics. *Proceedings of Online Educa Berlin*, 2006.

[11] S.W. Chan. *Routledge Encyclopedia of Translation Technology*. Taylor & Francis, 2014.

[12] The GF community. Grammatical Framework Demos. `http://www.grammaticalframework.org/demos/index.html`. (Accessed on 08/09/2017).

[13] Dana Dannélls, Mariana Damova, Ramona Enache, and Milen Chechev. Multilingual online generation from semantic web ontologies. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, pages 239–242, New York, NY, USA, 2012. ACM.

[14] DarkSky. The Dark Sky database. http://www.darksky.net/, 2017.

[15] Digital Grammars and the GF community. Demo: GF Wide Coverage Translation. `http://cloud.grammaticalframework.org/wc.html`. (Accessed on 08/09/2017).

[16] Digital Grammars and the GF community. GF Offline Translator - a mobile speech and text translation app for Android and iOS. `http://www.grammaticalframework.org/demos/app.html`. (Accessed on 08/09/2017).

[17] Digital Grammars and the GF community. GF Offline Translator – Android Apps on Google Play. `https://play.google.com/store/apps/details?id=org.grammaticalframework.ui.android`. (Accessed on 08/09/2017).

[18] Eva Forsbom. Rhetorical structure theory in natural language generation, Spring 2005.

[19] Thomas Hallgren. The correctness of insertion sort, 2001. Manuscript, Chalmers University.

[20] Kristofer Johannisson. *Formal and informal software specifications*. Citeseer, 2005.

[21] Gleb Lobanov. GitHub - gleblobanov/gf-mlg: The Weather Report in English and Russian. `https://github.com/gleblobanov/gf-mlg`. (Accessed on 07/31/2017).

[22] Gleb Lobanov and Krasimir Angelov. Planning for natural language generation in gf. In *Proceedings of Workshop on Logic and Algorithms in Computational Linguistics 2017 (LACompLing2017)*, 2017.

[23] William C Mann and Sandra A Thompson. Rhetorical structure theory: Toward a functional theory of text organization. *Text-Interdisciplinary Journal for the Study of Discourse*, 8(3):243–281, 1988.

[24] Per Martin-Löf. *Intuitionistic Type Theory*. Napoli: Bibliopolis, 1984.

[25] Per Martin-Löf. Mathematics of infinity. In Per Martin-Löf and Grigori Mints, editors, *Conference on Computer Logic*, volume 417 of *Lecture Notes in Computer Science*, pages 146–197. Springer, 1988.

[26] Bengt Nordström. Towards a theory of document structure. In Yves Bertot, Gerard Huet, Jean-Jacques Levy, and Gordon Plotkin, editors, *From Semantics to Computer Science: Essays in Honor of Gilles Kahn*, pages 265–279. Cambridge University Press, 2008.

[27] P Petri Mäenpää and Aarne Ranta. The type theory and type checker of gf. In *Colloquium on Principles, Logics, and Implementations of High-Level Programming Languages. Workshop on Logical Frameworks and Meta-languages, Paris*, 1999.

[28] François Portet, Ehud Reiter, Albert Gatt, Jim Hunter, Somayajulu Sripada, Yvonne Freer, and Cindy Sykes. Automatic generation of textual summaries from neonatal intensive care data. *Artificial Intelligence*, 173(7-8):789–816, 2009.

[29] Aarne Ranta. *Type-Theoretical Grammar*. Oxford University Press, 1994.

[30] Aarne Ranta. The GF resource grammar library. *Linguistic Issues in Language Technology*, 2009.

[31] Aarne Ranta. Grammatical Framework Tutorial. `http://www.grammaticalframework.org/doc/tutorial/gf-tutorial.html#toc18`, December 2010. (Accessed on 08/08/2017).

[32] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011. ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).

[33] Aarne Ranta. Molto: Multilingual on-line translation. FreeRBMT11, Barcelona, 2011.

[34] Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems*. Cambridge University Press, New York, NY, USA, 2000.

[35] Benoît Thouin. The meteo system. *Practical experience of machine translation*, pages 39–44, 1982.

[36] Canadian International Trade Tribunal. JOHN CHANDIOUX EXPERTS-CONSEILS INC. File Nos. PR-2001-029 and PR-2001-032, July 2002.