# Multilingual Text Generation for Abstract Wikipedia in Grammatical Framework: Prospects and Challenges

Aarne Ranta

**Abstract** Abstract Wikipedia is an initiative to produce Wikipedia articles from abstract knowledge representations with multilingual natural language generation (NLG) algorithms. Its goal is to make encyclopaedic content available with equal coverage in the languages of the world. This paper discusses the issues related to the project in terms of an experimental implementation in Grammatical Framework (GF). It shows how multilingual NLG can be organized into different abstraction levels that enable the sharing of code across languages and the division of labour between programmers and authors with different skill requirements. The plan is to start with a simple but functional multilingual NLG system and to proceed towards more and more sophisticated language and wider coverage of topics, also allowing a human in the loop to create content via a Controlled Natural Language (CNL).

**Key words:** Abstract Wikipedia, Controlled Natural Language, Grammatical Framework, Natural Language Generation, Text robots, Wikidata, Wikipedia

## 1 Introduction

Abstract Wikipedia is a recent initiative launched by the Wikimedia Foundation [44]. Its purpose is to support the universal availability of Wikipedia in different languages. At the time of writing, Wikipedia has 328 languages, most of which have only very few articles available[1].

Aarne Ranta
Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg
aarne.ranta@cse.gu.se

[1] See https://meta.wikimedia.org/wiki/List_of_Wikipedias for an up-to-date list.



To appear in Loukanova, R., Lumsdaine, P. L., and Muskens, R., editors, Logic and Algorithms in Computational Linguistics 2021 (LACompLing2021), Studies in Computational Intelligence (SCI), Springer Nature Switzerland, Cham, 2022.

The method chosen in Abstract Wikipedia is Natural Language Generation (NLG) based on Wikidata, which is a database of formalized facts [45]. These facts are stored as **RDF triples** (Resource Description Framework), which are two-place predications of the form $xRy$, where $R$ is a predicate and $x$ and $y$ are its arguments. As an example, consider the fact

```
wd:Q30 wdt:P1082 331449281
```

whose parts are the unique identifiers `wd:Q30` (for the USA) and the relation `wdt:P1082` (for the predicate "has population") and a numeric constant. Simple NLG can convert this triple to texts such as

*The population of the United States is 331,449,281.* (English)
*Yhdysvaltain asukasluku on 331 449 281.* (Finnish)
*La population des États-Unis est 331 449 281.* (French)
*Die Einwohnerzahl der Vereinigten Staaten ist 331.449.281.* (German)

The advantages of NLG, as opposed to manual authoring and translation, are several:

- **Consistency**: the content can be guaranteed to be the same in all versions of Wikipedia.
- **Speed**: versions in different languages can be produced in a fraction of a second.
- **Cost**: no human labour is needed to create new articles, but only for developing the algorithms.
- **Updates**: the content of articles can be kept up to date when facts change.
- **Customization**: different views of the same content, e.g. summaries and local adaptations, can be produced via suitable parameters in the algorithms.

All of these advantages are relevant for Wikipedia. Inconsistencies are a known problem in the current set-up based on manual work: versions in different languages can vary in size and content, and even contradict each other. Producing content in new languages is slow. Cost is one of the main reasons of this: it is labour-intensive to write and translate articles. Delayed updates cause errors in even well-supported languages. Customization is very limited, as it has to be done manually.

Now, NLG is an old idea with well-known algoritms [39]. It has even been used in Wikipedia, often under the name of **text robots**, which have produced millions of articles in several languages [46]. Hence, given the advantages of NLG, why is it not the standard way to produce and translate articles? Anyone who has tried to use NLG or read articles produced by text robots can probably list a number of problems that explain this:

- **Style**: NLG-produced text is "robotic" — boring, repetitive, unidiomatic.

- **Lack of data**: most of the content included in Wikipedia articles is not available in Wikidata or in any other database of formalized facts, and much of it might not even be possible to formalize.
- **Cost**: developing NLG algorithms might be a one-time cost, but so high that it is cheaper to write articles by hand.
- **Human resources**: writing NLG algorithms is a skill that might not be available for all languages.
- **Community resistance**: text robots, machine translation, and other automatic language processing methods are often discouraged or even prohibited in the Wikipedia community [47].

In this paper, we will outline an approach to Abstract Wikipedia that demonstrates the advantages and addresses the challenges. The presentation is based on actually existing code, which is publicly available[2]. However, as the code is a moving target under constant development, we will neither show all details of it here nor guarantee that the examples are completely up to date. The code repository itself contains updated documentation and also a tutorial for readers who want to try it out or develop it further.

## 2 From Templates to Rendering Functions

The simplest kind of NLG, often used in text robots, is **templates**: texts with slots for variable arguments. Thus the following template could express the population of any country (or other geographical area) in English:

```
The population of {X} is {Y}.
```

Templates work reasonably well in English, where words need not often be inflected. However, a familiar exception is shown when one of the variables is a number attached to a noun:

```
You have {X} new messages.
```

If $X$=1, the result is grammatically incorrect and reveals the robotic origin of the text. In other languages, the limits of templates are reached much more often. Thus in French, one should produce

*La population de la Suède* for $X$=Suède (Sweden),
*La population du Danemark* for $X$=Danemark (Denmark),
*La population des États-Unis* for $X$=États-Unis (United States).

The variable in the template could of course contain not only the country name but also the preposition with the article (*de la*, *du*, *des*). However, the country name is also used in other contexts, with other prepositions and possibly without articles. For example, to express "in a country", we write

[2] https://github.com/aarneranta/NLG-examples

*en Suède*,
*au Danemark*,
*aux États-Unis*.

Yet different forms are needed when the country name appears as a subject or an object of a sentence.

A purely template-based solution to the country name problem is to insert a "carrier noun", *pays* ("country"), which is inflected in a uniform way:

```
La population du pays {X} est {Y}.
{Y} habite dans le pays {X}.
```

(the latter means "$Y$ lives in the country $X$.") This technique is very commonly used in internet services, which do not always need to hide their robotic origin. But it is also used for rendering person profiles in social media. Since such media try to give a friendly impression, robotic language in them can be disturbing. In Wikipedia, it would result in a kind of text that the community, or readers in general, would have difficulties to accept.

A solution to the problem, proposed for Abstract Wikipedia, is to replace templates by proper **rendering functions**. Such a function could for instance wrap template variables by calls of grammatical case:

```
La population {GENITIVE(X)} est {Y}.
{Y} habite {LOCATIVE(X)}.
```

The system then needs, in addition to the templates, definitions of the `GENITIVE` and `LOCATIVE` functions for every possible value of $X$. This *can* be done as long as there is a limited number of such values, but it adds to the cost and the human resources needs of the system. Creating the templates also becomes more demanding, because the author needs to know where to use which of the cases. In fact, the problem is even more complex: think about facts of the form "$X$ was born in $Y$". The French template would need to make a difference between male and female values of $X$ to get the **agreement** of the word for "born" right:

```
{X} est {IF FEMININE(X) THEN née ELSE né} {LOCATIVE(Y)}
```

In addition to knowledge required about agreement in French grammar, the template notation itself starts to get complicated. And this is just the simplest case, with a choice from two forms: if the changing part is a verb, the template has to choose from six forms (two numbers times three persons).

As one more problem to be solved in rendering functions, even the order of words may need to be varied. Consider the German template for "$X$ was born in $Y$", for simplicity without cases (of which German has four):

```
{X} wurde in {Y} geboren.
```

Now, in Wikipedia, it is customary to indicate the sources of facts by links or references. When a fact is disputed, it can be appropriate to describe different opinions by phrases such as *nach Z* ("according to *Z*"), *Z glaubt, dass* ("*Z* believes that"). In such contexts, German grammar requires the word order of the "born" template to be changed:

```
Nach {Z} wurde {X} in {Y} geboren.
{Z} glaubt, dass {X} in {Y} geboren wurde.
```

Hence, not only do we need templates for thousands of predicates, but there have to be (at least) three templates for every predicate to get the word order right in all contexts, plus a device in the template notation that enables us to select the correct alternative.

## 3 Rendering Functions in Grammatical Framework

Agreement and word order are familiar from school grammar and hence by no means advanced concepts. But their precise treatment in formal grammars in computational linguistics is considered specialist knowledge, and in real-world NLG templates, they are usually avoided altogether by using techniques such as carrier nouns.

Grammatical Framework (GF, [34]) is a programming language that aims to make linguistic knowledge accessible to programmers. Abstract Wikipedia has mentioned GF as a possible technology, and the goal here is to investigate how far it can reach. In GF, grammatical features such as inflection, agreement, and word order can be defined by linguistically knowledgeable programmers in the form of software libraries and reused by non-linguist engineers for different purposes such as NLG [32]. In this way, an equivalent of linguistics-aware rendering functions can be written in a format that essentially looks as templates (see Sect. 6 below).

GF is a special-purpose functional programming language with many features, which can be learned from tutorials and manuals on the web[3] and from the GF book [34]. It inherits much of its syntax from Haskell[4], but adds some constructs relevant for grammars, such as regular expression pattern matching used for morphology. Like Haskell, GF is statically typed, which is a guarantee that grammars do not fail at runtime. A more special feature is **reversibility**: GF grammars can be used for both parsing and generation, as well as their composition, translation.

Viewed as a grammar formalism, a special feature of GF is that it divides grammar specifications into **abstract syntax** and **concrete syntax** parts. An abstract syntax defines a set of **trees**, and a concrete syntax specifies

[3] http://www.grammaticalframework.org/

[4] https://www.haskell.org/

how they are **linearized** in different languages. One and the same abstract syntax can have several concrete syntaxes, which results in **multilingual grammars**. In a multilingual grammar, **translation** is defined as parsing with one concrete syntax and linearization with another one.

The largest set of languages covered by a GF grammar known to us has over 90 languages and defines their numeral systems with a shared abstract syntax [19]. A more general grammar, the GF Resource Grammar Library, defines syntactic structure, morphology, and basic lexicon for 55 languages, of which around 40 have complete implementations of a comprehensive shared abstract syntax [33]. This library has played a major role in almost all multilingual text generation projects in GF, including academic projects on topics such as software specifications [7], mathematics [40, 35], cultural heritage [11], law [2], and healthcare [26], as well as various commercial projects[5].

As a first example of GF, consider templates of the form

```
the {F} of {X} is {Y}
```

which can express many RDF triples of Wikidata. Its implementation in GF consists of an **abstract syntax function** and its **linearization function**, marked by the keywords `fun` and `lin`, respectively:

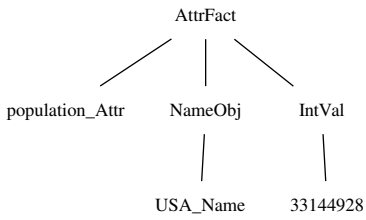
```
fun AttrFact : Attr -> Obj -> Val -> Fact
lin AttrFact attr obj val =
    "the" ++ attr ++ "of" ++ obj ++ "is" ++ val
```

In words, the function `AttrFact` takes three arguments — an attribute, and object, and a value — and returns a fact. Its linearization combines these arguments by concatenation (operator `++`), with some string literals added in between. The types of these arguments must match the argument types specified in the `fun` definition. Like Haskell, GF uses the arrow syntax for function types, and the prefix notation for function application.

The abstract function `AttrFact` can be used for building infinitely many trees of type `Fact`. An example is the tree

```
AttrFact population_Attr (NameObj USA_Name) (IntVal 331449281)
```

in GF's prefix notation. An equivalent graphical representation is




[5] Some of them can be traced from http://grammaticalframework.org

Given obvious linearization functions for `population_Attr`, `NameObj`, `USA_Name`, and `IntVal`, this tree is linearized to

*the population of the United States is 33144928.*

The above kind of purely concatenative linearization is a special case of GF, corresponding to templates with slots. Similar linearization functions could be defined for other languages as well — but, as we saw in Sect. 1, this would not generalize well over languages. To enable grammatically correct linearizations in all contexts and in all languages, GF generalizes linearization from string concatenation by adding three concepts: **parameters**, **tables**, and **records**.

Grammatical number, case, and gender are examples of parameters, defined as enumerated (and, more generally, finite algebraic) datatypes with the keyword `param`. The following definitions are suitable for German:

```
param Number = Sg | Pl
param Case = Nom | Acc | Dat | Gen
param Gender = Masc | Fem | Neutr
```

These definitions are used in type checking to guarantee the consistency of grammars. But unlike `fun` definitions, they belong to the concrete syntax: different languages can define Number, Case, and Gender in different ways — or not at all, if the language lacks the feature in question.

GF tables are used for expressing inflection tables such as noun declensions. Technically, they are functions over parameter types. As an example, consider the table for the German noun *Stadt* ("city") and its encoding in GF, in Fig. 1. Notice the **wildcard patterns** `_` that match cases that are not mentioned explicitly, with a standard notation used in functional programming. Wildcard patterns make it possible to avoid the repetition of similar forms, which typically appear in inflection tables written in the full form. The type checker of GF makes sure that all of the four cases and two numbers are matched.

In addition to the table, Fig. 1 indicates that *Stadt* is a feminine noun (n.f.). The gender of nouns is not a **variable feature** that produces different forms like number and case do (with some exceptions such as *König - Königin* "king - queen"): it is an **inherent feature** of nouns. Inherent features can be collected into GF records, together with all other information about a word. The record shown in Fig. 1 contains both the inflection table and the inherent gender.

Morphological features are used in syntax to implement **agreement**. To give one example in full detail, let us consider a simple one: *one item* vs. *two items*. A possible abstract syntax function for *you have X Ys* is

```
fun YouHaveItems : Numeral -> Item -> Statement
```

*Stadt*, n.f.

| | **Sg** | **Pl** |
|---|---|---|
| Nom | *Stadt* | *Städte* |
| Acc | *Stadt* | *Städte* |
| Dat | *Stadt* | *Städte* |
| Gen | *Stadt* | *Städten* |

```
{s = table {
    Sg => table {_ => "Stadt"} ;
    Pl => table {Dat => "Städten" ; _ => "Städte"} ;
 g = Fem
}
```

**Fig. 1** Inflection table and inherent gender of German *Stadt* and its representation in GF.

A proper linearization requires that every `Numeral` has a grammatical number (singular or plural) attached as an inherent feature. For example[6],

```
lin one_Numeral = {s = "one" ; n = Sg}
lin two_Numeral = {s = "two" ; n = Pl}
```

In addition, every `Item` (noun) has grammatical number as a variable feature, producing different forms in a table:

```
lin message_Item = table {Sg => "message" ; Pl => "messages"}
```

The linearization function of `YouHaveItems` implements agreement as an interplay between inherent and variable features:

```
lin YouHaveItems num it = "you have" ++ num.s ++ it!num.n
```

Here, `num.s` is the `s`-field projected from the record, and `it!num.n` is the value selected from the table to match the `n` field of the record.

## 4 Abstraction Levels in GF

The simple mechanism of tables and records has turned out sufficient to model all kinds of agreement and other variation found in the languages that have been implemented in GF so far, including, in addition to several Germanic, Romance, and Slavic languages, also Fenno-Ugric [25], Indo-Iranian [43], Semitic [10, 8], Bantu [30, 5, 22], and East-Asian [48, 38] languages.

To give one more example, German word order can be defined by a table that reorders the subject, verb, and complement, as a function of a parameter

[6] These are special cases of a more general recursive definition of numerals [19]. Different languages may require more distinctions than just singular/plural: Arabic, for instance, has five different agreement patterns, whose GF implementation is explained in [9]

that stands for main clause, inverted clause, or subordinate clause (which are three values of an Order parameter type):

```
table {
  Main => subj ++ verb ++ compl ;
  Inv => verb ++ subj ++ compl ;
  Sub => subj ++ compl ++ verb
  }
```

At least as important for the current task as the record and table mechanism itself is the possibility to hide it. This is provided by the **Resource Grammar Library** (RGL), which defines the details of syntax and morphology and exports them via a high-level API (Application Programming Interface) [33]. The complete API is at the time of writing available for 40 languages. It also contains extensive lexical resources for more than half of them. Table 1 relates the available RGL resources to Wikipedias in different languages.

With the RGL API, the linearization of `YouHaveItems` can be defined as follows:

```
lin YouHaveItems num it = mkCl you_NP have_V2 (mkNP num it)
```

The API function `mkCl` builds a clause (`Cl`) from a noun phrase (`NP`), a two-place verb (`V2`), and another noun phrase[7]. The pronoun `you_NP` is directly available in the API, and so is the verb `have_V2`[8]. The object noun phrase is built by the function `mkNP`, which combines a numeral with a noun. Under the hood, `mkNP` takes care of the choice of the singular or the plural, whereas `mkCl` takes care of subject-verb agreement (*have* vs. *has*)[9].

So far, we have seen how GF can express agreement and other kinds of variation and how the RGL can hide the details from the application programmer. But we have not seen the ultimate abstraction yet: the language-independence of the RGL API. The above linearization rule for `YouHaveItems` has in fact *exactly the same code* for every language that implements the RGL API, but compiles into different records and tables under the hood. To give some examples of what happens,

- `mkNP` selects the gender of the numeral as a function of the noun in Romance and Slavic languages,

[7] It is a convention in the RGL that syntactic functions building values of type $C$ have the name `mk`$C$, i.e. "make" $C$. These names are overloaded: all such functions can have the same name, as long as they have different lists of argument types so that the type checker can resolve them. Thus it is often possible to guess a function name without looking it up.

[8] These functions, as lexical functions in general, take no arguments and cannot thus be resolved by type checking. The RGL convention is to denote them by English words suffixed with part of speech tags.

[9] The concept of a verb in the RGL is more abstract than in traditional grammars. Thus the abstract two-place verb `have_V2` is in some languages implemented with non-verbal constructions such as prepositional phrases in Arabic: "I have a dog" is rendered *ladayya kalbun*, literally "with me a dog".

| language | articles | RGL | language | articles | RGL |
|---|---|---|---|---|---|
| English | 6,545,975 | +++ | Esperanto | 323,608 | + |
| Cebuano | 6,125,812 | - | Hebrew | 321,316 | + |
| German | 2,719,877 | +++ | Danish | 284,290 | ++ |
| Swedish | 2,552,522 | +++ | Bulgarian | 283,953 | +++ |
| French | 2,450,741 | +++ | Slovak | 241,847 | + |
| Dutch | 2,099,691 | +++ | Estonian | 229,915 | +++ |
| Russian | 1,849,325 | +++ | Greek | 212,862 | ++ |
| Spanish | 1,798,346 | +++ | Lithuanian | 204,111 | + |
| Italian | 1,769,757 | +++ | Slovenian | 177,533 | +* |
| Egyptian Arabic | 1,597,544 | ** | Urdu | 176,166 | +++ |
| Polish | 1,534,113 | +++ | Norwegian (Nynorsk) | 162,695 | ++ |
| Japanese | 1,340,051 | ++ | Hindi | 152,475 | +++ |
| Chinese | 1,300,293 | +++ | Thai | 149,693 | +++ |
| Vietnamese | 1,275,688 | - | Tamil | 148,547 | + |
| Waray-Waray | 1,265,938 | - | Latin | 136,958 | +* |
| Ukrainian | 1,190,703 | - | Latvian | 115,349 | +++ |
| Arabic | 1,184,349 | ++ | Afrikaans | 104,596 | +++ |
| Portuguese | 1,094,514 | +++ | Swahili | 74,639 | +* |
| Persian | 925,446 | ++ | Icelandic | 54,803 | ++ |
| Catalan | 709,317 | +++ | Punjabi | 38,549 | ++ |
| Serbian | 662,099 | - | Nepali | 32,241 | ++ |
| Indonesian | 627,502 | * | Interlingua | 24,231 | ++ |
| Korean | 603,549 | +* | Mongolian | 21,436 | +++ |
| Norwegian (Bokmål) | 597,046 | ++ | Sindhi | 15,251 | ++ |
| Finnish | 537,889 | +++ | Amharic | 15,051 | ++ |
| Turkish | 514,410 | +* | Zulu | 10,583 | + |
| Hungarian | 511,089 | + | Somali | 8,467 | +* |
| Czech | 509,155 | + | Maltese | 4,842 | +++ |
| Chechen | 481,958 | - | Xhosa | 1,240 | + |
| Serbo-Croatian | 456,901 | + | Tswana | 773 | + |
| Romanian | 433,112 | +++ | Greenlandic | 244 | + |
| Min Nan | 431,714 | - | Greek (Ancient) | - | + |
| Tatar | 417,595 | - | Chiga (Rukiga) | - | + |
| Basque | 397,843 | ++ | Kikamba | - | + |
| Malay | 360,146 | + | Egekusii | - | + |

**Table 1** Wikipedias in different languages, according to https://meta.wikimedia.org/wiki/List_of_Wikipedias retrieved 30 August 2022, and their coverage in the current GF RGL, sorted by the number of articles. The third column shows the status of the RGL: +++ means full API coverage with a large lexicon, ++ means full or almost full API coverage with a smaller lexicon, +* partial API coverage with a large lexicon, + means implementation started, - means not started. * and ** mean corresponding coverage in a closely related language: Arabic for Egyptian Arabic, Malay for Indonesian. The languages on the left are the 35 top languages of Wikipedia, including those without RGL. The languages on the right are those after the top 35 that have some RGL coverage. Four of the RGL languages were not found in the list of Wikipedias, but this may be due to different names used for them.

- `mkNP` selects the number and case of the noun as a function of the number in Slavic languages and Arabic,
- `mkNP` adds a classifier to the noun in Chinese and Thai,
- `mkCl` selects the word order in German, Dutch, and Scandinavian,
- `mkCl` selects complement cases and prepositions of two-place verbs as needed in almost all languages,
- `mkCl` implements the ergative agreement in Basque, Hindi, and Urdu,
- `mkCl` selects and orders clitic pronouns in Romance languages and Greek.

To sum up, using RGL API implies that

- those who write rendering functions do not need to worry about low-level linguistic details, but only about the abstract syntax types of their arguments and values;
- a rendering function written for one language is ready to be used for all RGL languages.

These two things together have made GF into a productive tool for multilingual NLG. The code sharing for rendering functions is formally implemented by means of **functors**, a.k.a. **parameterized modules**, which are instantiated to different languages by selecting different instances of the RGL API [31]. In this way, the actual code for the rendering functions does not even need to be seen by the programmer that uses the code for a new language.

Scaling up GF and RGL to the Wikipedia task requires "only" that the RGL be ported to all remaining Wikipedia languages. What this means in terms of effort and skill is a topic worth its own discussion, to which we will return in Sect. 10.

## 5 Smart Paradigms and the Lexicon

The RGL API offers functions such as `mkCl` and `mkNP` shown above to combine phrases into larger phrases. The smallest building blocks of phrases are **lexical units**, i.e., words with their inflection tables and inherent features.

We have seen *one*, *two*, and *message*, as examples of lexical units represented as records and tables. The RGL provides high-level APIs for constructing them. For most languages, it provides **smart paradigms**, which build complete tables and records from one or few characteristic forms [12]. For example, English has two smart paradigms for nouns (`N`):

```
mkN : Str -> N
mkN : Str -> Str -> N
```

The former paradigm takes one string as its argument, the singular form of the noun. It returns a table that also contains the plural form, where the usual stem alternations are carried out, such as *baby-babies*, *bus-buses*. To

form the plural, **regular expression pattern matching** is used. A slightly simplified function for this, also usable for the 3rd person singular present indicative of verbs, is

```
add_s w = case w of {
  _ + ("a"|"e"|"o"|"u") + "y" => w + "s" ;      -- boy, boys
  stem + "y"                  => stem + "ies" ; -- fly, flies
  _ + ("ch"|"s"|"sh"|"x"|"z") => w + "es" ;     -- bus, buses
  _                           => w + "s"        -- cat, cats
  }
```

(the notation `--` marks comments in GF, here showing examples of words matching each pattern). If a noun has an irregular plural (like *man-men*), the two-argument function is used. Thus the programmer who builds a lexicon just needs to use expressions such as

```
mkN "continent"
mkN "country"
mkN "Frenchman" "Frenchmen"
```

with no worries about the internal records and tables or pattern matching. In German, a particularly useful paradigm is

```
mkN : Str -> Str -> Gender -> N
```

which can for instance generate the record shown in Fig. 1 above, by

```
mkN "Stadt" "Städte" Fem
```

For languages other than English, smart paradigms typically have to do more work, such as produce the 51 forms of the French verb or over 200 forms of the Finnish verb. Evaluations have shown that even in highly inflected languages, all forms of most words can be inferred from just one or two characteristic forms [12][10]. This means that a morphologically complete lexicon can be built rapidly and on a low level of skill. What is more, lexicon building can often be automated: a list of words equipped with part of speech information (noun, adjective, verb) can be mechanically converted into a list of smart paradigm applications.

Many languages have existing morphological dictionaries independent of GF, for instance in Wiktionary as well as in Wikidata itself[11]. Such resources can often be converted to GF records and tables, which means that rendering functions can just use abstract syntax names such as `country_N` and not even care about smart paradigms. However, previously unseen words can always be encountered in texts, especially ones containing specialized terminology, and

[10] A typical exception are Indo-European verbs that may need three or more forms, but there are usually just a few hundred of them, and they can be collected into a static lexicon.

[11] Wikidata lexicographical resources, https://www.wikidata.org/wiki/Wikidata:Lexicographical_data

smart paradigms are then needed to add them to the lexicon. Moreover, for many of those 300 languages that Abstract Wikipedia targets, comprehensive morphological dictionaries do not exist, and defining smart paradigms for them is an essential part of the RGL building effort.

The lexical items standing for atomic concepts are often not expressible by single words: depending on language, they may be **multiword expressions** with internal syntactic structure. As an example, consider the concept "standard data protection clause" from the General Data Protection Regulation (GDPR) of the European Union[12]. It is a typical example of a legal concept that has established translations into different languages. Its syntactic category is common noun (CN), equipped with a plural form and, in many languages, a gender. To form the plural and identify the gender, one needs to know the syntactic structure — in particular, the **head** of the phrase. Thus we have:

- *standard data protection clause(s)* (English, head last),
- *clause(s) type(s) de protection des données* (French, head first, first two words inflecting),
- *clausol(a/e) tipo di protezione dei dati* (Italian, head first, only the first word inflecting),
- *Standarddatenschutzklausel(n)* (German, single word).

For multiword terms, a handful of syntactic functions are needed in addition to the morphological paradigms in order to define inflection and gender. Thus for instance the French linearization is defined by adding layers of modification to the head noun `clause_N`:

```
mkCN (mkCN type_A clause_N)
  (mkAdv de_Prep (mkNP (mkCN protection_N
    (mkAdv (mkAdv de_Prep (mkNP thePl_Det donnée_N))))))
```

Writing such complex GF expressions by hand can be demanding, but they can fortunately often be obtained by using RGL grammars for **parsing**. In the present case, the parser must convert the string *clause type de protection des données* into an abstract syntax tree of type CN. This technique, known as **example-based grammar writing**, has been used to enable native speakers to provide grammar rules without writing any code [37].

## 6 More Abstraction Levels

We have gone through three abstraction levels that a GF rendering function (i.e. linearization) can be defined on:

[12] Over 3000 GDPR concepts in five languages have been collected to a GF lexicon in a commercial project, https://gdprlexicon.com/.

- Records and tables, mostly needed just inside the RGL;
- RGL API functions, used for building new rendering functions;
- Wikipedia rendering functions, such as `AttrFact`.

Higher levels can use the lower levels as **libraries** (in the sense of software libraries), which means that they can take earlier work for granted. As explained in more detail in Sect. 10, we do not expect Abstract Wikipedia authors to use GF on the level of records and tables. Even the level of RGL API is too low for most authors: the main level to work on will be by using the high-level rendering functions, built by a smaller group of experts.

On the level that uses Wikipedia rendering functions, only a small fragment of GF notation is needed: function applications and strings. These constructs are so ubiquitous that they do not even need the GF programming language. Instead, they can be used directly in a general purpose language via **bindings** that are available as a part of the GF software. This technique is called **embedded grammars** and enables programmers to use GF grammars without writing any GF code.

To give an example, consider a rendering function for "the $F$ of $X$ is $Y$" and its RGL linearization for different languages,

```
fun AttrFact : Attr -> Obj -> Val -> Fact
lin AttrFact attr obj val =
  mkCl (mkNP the_Det (mkCN attr obj)) val
```

A grammar module containing this function can be called from Python by importing it as a Python module, calling it (for example) `G`, and writing

```
G.AttrFact(G.population_Attr, G.NameObj(name), G.IntVal(pop))
```

where the variables `name` and `pop` get their values directly from Wikidata. The only API that the programmer needs to know are the abstract syntax types of the linearization functions.

Embedded GF grammars are available for C, C#, Haskell, Java, and Python[13]. A further abstraction level on top of this is **Wikifunctions**, which is an emerging technology for accessing all kinds of functions via a web API, hiding the underlying programming language [44]. The plans for Abstract Wikipedia include making GF rendering functions available on this level.

Another direction in which the abstraction from GF code can be extended is by using the parser of GF from a general purpose language. The above call of the rendering function can thus be accessed via its linearization, where slots are left for the values. Here is the equivalent code in Python:

```
str2exp("the population of {X} is {Y}".format(X=name, Y=pop))
```

The string argument of `str2exp` looks exactly like a template, but `str2exp` calls the GF parser to convert strings to GF abstract syntax trees that can

[13] http://www.grammaticalframework.org/doc/runtime-api.html

be linearized in multiple languages, of course obeying all their grammatical rules beyond string concatenation.

With the parser, the programmer who implements NLG rules can thus use GF rendering functions without even knowing the names of those functions. She just needs to know what can be parsed by the grammar. However, since this can be difficult and error-prone, and since parsing can be ambiguous, the more precise use of imported modules may still be needed as a back-up.

## 7 Improving the Style

In classical rule-based NLG, data is converted to text in several steps [39]:

- Content determination: what to say.
- Text planning: in what order to say it.
- Aggregation: merging atomic facts into compound sentences.
- Lexical choice: selecting words for concepts.
- Referring expression generation: using pronouns and other alternative expressions for entities.
- Surface realization: selecting the final word order and morphological forms.

We have by now mostly focused on surface realization. When working on highly multilingual tasks, this is the most demanding component, because of the huge differences between the surface grammars of languages.. We have shown how the RGL gives a solution to this problem. Surface realization is also the clearest contribution that GF itself can make to NLG; most of the other steps can be easier to perform in a general purpose language embedding GF in the way shown in Sec. 6. These steps can operate on the abstract syntax of the GF and thereby deal with several languages at the same time.

Lexical choice is also defined by GF functions and their linearizations for all data entities. One improvement above the monotonic phrasing *the F of X is Y* is to define predicate-specific functions, such as

```
fun PopulationFact : Obj -> Int -> Fact
```

linearized *X has Y inhabitants*. Such functions can be language-specific, if a language happens to have a nice idiom for a certain concept. But they can also be cross-lingual and defined by functors, possibly for a subset of languages for which they are natural and which can implement them. Thus a new language added to the system can start with baseline, monotonic renderings early in the process and get incremental stylistic improvements later.

Starting with the beginning of the pipeline, content determination at its simplest is to take all facts in Wikidata about some object, such as a country, and convert them into sentences of the form "the $F$ of $X$ is $Y$", rendered as "the $F$s of $X$ are $Y$" when $Y$ contains multiple values. The resulting

text is extremely boring, but it "does the job" in the sense of expressing the information in a grammatically correct way in multiple languages.

Predicate-specific rendering functions are perhaps the simplest way to improve style. A more general way is to add functions that implement text planning, aggregation, and referring expression generation:

```
OneSentDoc : Sent -> Doc          -- S.
AddSentDoc : Doc  -> Sent -> Doc  -- D. S.
ConjSent   : Sent -> Sent -> Sent -- S and S
NameObj    : Name -> Obj          -- Argentina
PronObj    : Name -> Obj          -- it
```

These functions can be used to implement document-level templates, such as the following creating small but reasonably fluent articles about countries:

```
str2exp("Doc",
    ("{c} is a {co} country with {p} inhabitants . "
     "its area is {a} . "
     "the capital of {c} is {ca} and its currency is {cu} ."
     ).format(
         c=countr, co=cont, p=pop, a=area, ca=cap, cu=curr))
```

Our experimental implementation[14] has around 40 GF functions, which can be combined to text templates for different purposes; the main test cases have been geographical data and Nobel prize winners[15].

While the GF functions and templates are language-independent, the NLG system can customize their use for individual languages. As an obvious example, the area of a country may be converted from square kilometres to square miles for some countries. As a more intricate one, the referring expression generation may utilize the gender systems of different languages to enable the most compact expressions: in English, the pronoun *it* is often ambiguous and therefore not adequate, whereas each of German *er*, *sie*, *es* can be unambiguous in the same context.

## 8 Selecting Content

The document template for countries in the previous section builds a text from atomic facts: continent, population, area, capital, currency. All these facts are directly available as RDF triples in Wikidata and can therefore be automatically picked into documents. However, texts in general can choose to

[14] https://github.com/aarneranta/NLG-examples

[15] Notice that the `co` argument expects an adjective such as *Asian*. Such adjectives, known as demonyms, are in a natural way included in the linearization records of geographical names. Also notice that the string has been manually tokenized to help the GF parser.

drop out some facts and also to state facts that are not directly available in the data. For example, *China has the largest population in the world* is a fact verifiable in Wikidata, but requires a more complex query than an individual triple, for instance, a Python expression

```
maxpop = max(cont_data, key=lambda c: c.population).country
```

which is an example of **aggregation** in the database (rather than NLG) sense. Once this query has been performed and the fact established, the value of `maxpop` can be reported in a text, instead of stating the exact populations of all countries.

Here is a Python template for summaries of facts about continents and the whole world:

```
doc = factsys.str2exp("Doc",
    ("there are {n} countries in {co} ."
     "the total population of {co} is {p} ."
     "{mp} has the largest population "
     "and {ma} has the largest area .".format(
       n = ncountries, co = cont, p = totalpop,
       mp = maxpop, ma = maxarea)
```

Here is a text generated in English, Finnish, and German. The last sentence of these texts is not shown in the template. It is included only in regions that actually have countries with over a billion inhabitants[16].

*There are 194 countries in the world. The total population of the world is 7552 million. China has the largest population and Russia has the largest area. India and China are the only countries with over a billion inhabitants.*

*Maailmassa on 194 maata. Maailman yhteenlaskettu asukasluku on 7552 miljoonaa. Kiinalla on suurin asukasluku ja Venäjällä on suurin pinta-ala. Intia ja Kiina ovat ainoat maat, joissa on yli miljardi asukasta.*

*Es gibt 194 Länder in der Welt. Die gesamte Einwohnerzahl der Welt ist 7552 Millionen. China hat die größte Einwohnerzahl und Russland hat die größte Fläche. Indien und China sind die einzigen Länder mit über einer Milliarde Einwohnern.*

Notice that all the facts stated in the above texts may change over time. For example, India is predicted soon to have a larger population than China. A new text can then be generated by exactly the same grammar, template, and queries, to reflect this change.

[16] The expression *7552 million* is a result of rounding the exact population. Different rounding functions are available in the grammar, and the NLG has the task to select an approriate one.

## 9 Authoring

In the previous examples, as usual in traditional NLG, the content was selected automatically by an algorithm that decided which facts where interesting. The algorithm also decided the text structure. This is undeniably the fastest and cheapest way to make information available in natural language.

However, fully automatic NLG has two serious shortcomings. First, algorithms cannot always predict which facts are interesting for human readers. Secondly, the content in Wikipedia articles is not always available in Wikidata. To solve these problems, we invoke the parser of GF once again and move from automated NLG to **interactive authoring**. The idea is simply that articles are written by human authors, just like in traditional Wikipedia. But unlike in traditional practices, they are parsed into abstract syntax, verified with respect to Wikidata, and automatically translated to all languages that have a concrete syntax in GF. What is more, the document that is stored as abstract syntax can be edited later by other authors, possibly via input in other languages. Figure 2 shows the architecture of the authoring system.

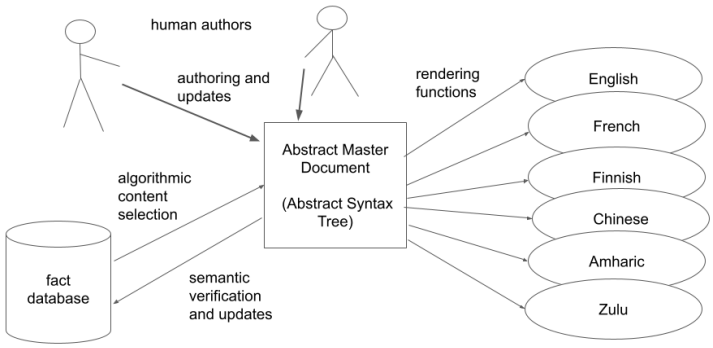



**Fig. 2** Text generation from a fact database with a human author in the loop.

The idea of interactive authoring was in fact the starting point of whole GF. GF was first created at Xerox in the project entitled Multilingual Document Authoring [14], which in turn was inspited by WYSIWYM ("What You See is What You Mean", [29]). GF was designed to be a general formalism for defining Controlled Natural Languages (CNL). A CNL is a fragment of natural language with (often, even if not always) formal grammar and semantics [23, 16]. In the case of GF, the formal grammar is moreover divided into abstract and concrete parts, which makes it multilingual [4].

One challenge in CNL-based authoring is to make sure that the authors' input can be parsed in the fragment recognized by the grammar. Tools for this have been developed throughout the history of GF [18, 21, 36, 20, 1], but there is still room for improvement.

Another challenge, more specific to the Wikipedia case, is that the formal semantics is never going to be complete. As already mentioned, many things that authors want to express might not be present in Wikidata. It can also happen that Wikidata is wrong, or even contradictory. The authoring system can accommodate this by enabling the author to extend or change Wikidata when such situations occur.

## 10 Roles and Skills

The success of Abstract Wikipedia depends ultimately on getting the community involved. No person or research group could ever recreate all the content in abstract form and define all the required rendering functions in all languages. Community members should be enabled to contribute at different levels of skills:

- GF implementers: a specialist group working on the internal algorithms of GF and their integration with Wikipedia software;
- RGL writers: linguistically proficient programmers implementing new languages;
- NLG programmers: implementers of rendering functions for new concepts and new textual forms;
- Domain experts: programmers who adapt rendering functions to new domains and terminologies;
- Content creators: authors of articles and providers of Wikidata.

As we go down the list, more and more people are needed in each role, while their GF and programming skill requirements decrease.

Starting with GF implementers, this has always been a group of just a few persons. We do not expect that many more are necessarily needed, because GF is already a mature and stable software infrastructure that has been used in numerous NLG projects. Some new functionalities are planned as research projects, in particular dynamic loading and updating of grammars, automated extraction of parts of grammars, and support for Wikifunctions. But even the current implementation is sufficient to get started.

RGL writers are needed for every new language, one or two per language, which means that a few hundred need to be trained to reach the goal of 300 languages. The series of GF Summer Schools organized since 2009[17] has shown that a two-week intensive training is enough for this task. Writing an RGL implementation for a new language itself typically takes two to six person months. In the Abstract Wikipedia project, this task can, however, be started with a subset of the RGL: the experimental implementation described here uses less than 20% of the common abstract syntax functions. Hence a

[17] http://school.grammaticalframework.org

basic rendering system for a new language can be released after a few weeks' work. By using functors (see Sect. 4), one can moreover share most of the concrete syntax code inside language families, so that only one language in the family needs a fully procifient RGL programmer, whereas the others can be added more easily by informants for those languages who provide values to functor parameters. This method has been extensively applied in Scandinavian, Romance [31], Indo-Iranian [43], and Bantu [22] families.

NLG programmers implement rendering functions by using the RGL, as well as text templates written in a general purpose host language such as Python. These programmers need to have skills in the host language but can do with a fragment of GF sufficient for using the RGL API.

Domain experts have as their main task to define rendering functions for technical terms, that is, lexical items and multiword expressions. They do not necessarily need GF knowledge, but can just provide strings to be parsed and converted to records and tables via morphological lexica and RGL functions, including smart paradigms. In other words, the domain expert provides the translations of the terms, whereas the RGL generalizes them to rendering rules. The process can be helped by existing sources such as Wikidata lexical resources and WordNets [15, 3].

Content creators, finally, will be the vast majority of Abstract Wikipedia contributors. All Wikipedia authors should be enabled to create abstract content via an easy to use authoring system. Ideally, the overhead of generating abstract content should be small enough to be compensated by the availability of automatic translations to other languages and semantic checking.

## 11 First Results

Most of the examples above come from an experiment where Wikidata about 194 countries was used for generating articles in four languages: English, Finnish, German, and Swedish. To show how the methods generalize, the experiment was completed by articles about 818 Nobel prize winners:

> *Gerty Cori won the Nobel Prize in Physiology or Medicine in 1947. She was born in Prague in the Czech Republic in 1896. Cori died in 1957. Gerty Cori was the first woman that won the Nobel Prize in Physiology or Medicine.*

The main new feature shown here is the use of tenses. Apart from that, the rendering functions originally written for geographic data were sufficient for the new domain with only a few additions[18].

The first experiment was carried out by the author of this paper. But how easy is the task for people with no previous GF experience? This was tested

[18] One problem should be noticed: in 1896, the Czech Republic did not exist, but Prague was a city in the Austro-Hungarian Empire. Thus more care would have been needed when combining Nobel prize data with geographical data.

in Spring term 2022 by a group of six Bachelor students in Computer Science, who built a system addressing 1235 localities in Sweden [13]. For this group, a 4-hour crash course in GF was given, as well as weekly 1-hour supervision sessions during 12 weeks. The group used the same basic grammar as in the original experiment. In addition to extending the lexicon in the domain expert role, they also extended the texts with new kinds of information, for example, about famous persons and their professions coming from the localities. This experiment can be considered successful in showing how much can be done with a minimal specific training.

Four other student projects in Spring term 2022 were Masters theses addressing research questions related to Wikipedia. One was about formal semantics, interpreting abstract syntax trees in a model based on Wikidata [41]. This was a standard kind of Montague-style logical semantics [27, 42] extended with anaphora resolution and a treatment of sentences that could not be decided in the model. What results from the latter is technically a third truth value. The authoring interface can warn the user about it, or also give her the option to update the semantic model with new facts.

A second Masters thesis built language models, both n-gram and neural, for Wikipedia in 46 languages, with the purpose to assess the fluency of generated texts and help select the most fluent ones of alternative renderings in each language [24]. Unlike purely statistical or neural generation, which has no guarantees to match a non-linguistic reality [6], this use of language models is controlled by the semantics. In the resulting pipeline, rule-based NLG generates several renderings of the same facts, all of them semantically correct, and the language model ranks them in terms of fluency.

The third project was about multilingual treebanks built by parsing a set of Wikipedia articles in 58 languages with Universal Dependencies (UD, [28]). The result was a ranked set of syntactic structures used in Wikipedias, as well as a cross-lingual comparison of how similar the structures are [17]. This makes it possible to estimate how often a functor-based linearization is adequate. The results were moreover used in a fourth project, still in progress at the time of writing, to extract GF rendering functions from UD trees that are actually used in the current Wikipedia.

In addition to the directly NLG-related projects, resource development has started to export GF lexical data into Wikidata, in connection with the GF-WordNet project[19]. At the time of writing, such resources have been exported for 24 languages, around 590k entries per language.

[19] http://cloud.grammaticalframework.org/wordnet/

## 12 Conclusion

The goal of this paper has been to show that the advantages of NLG — consistency, speed, cost, updates, and customization — are relevant for Wikipedia and can be scaled up to a multilingual setting by using GF. We have gone through some results from an experimental implementation and addressed some known problems and how to mitigate them:

- Robotic style: can be improved by adding more NLG functions (Sect. 7) and, in particular, by enabling human authoring via a CNL (Sect. 9) .
- Lack of data: can be completed as a by-product of authoring (Sect. 9). But even now, there is enough Wikidata to create useful content even though the whole Wikipedia cannot be covered.
- Cost: can be reduced by abstract syntax and functors, which enable code sharing between languages (Sect. 3).
- Human resources: can be helped with tools such as parsing and authoring, so that most of the work needs only minimal training (Sect. 10) .
- Community resistance: many of the earlier problems of text robots and automatic translation can be avoided. But it remains to be seen how the wider community responds to this initiative.

A baseline implementation already exists and can be applied to new areas of knowledge and adapted to new languages. Even though perfection might never be achieved, gradual improvements can enable more and more content to be created with better and better style in more and more languages.

Using GF for this task has a number of advantages. First, the Resource Grammar Library is a unique resource for multilingual generation functions. As it contains even less-represented languages (see Table 1), adding more such languages looks like a realistic goal.

Secondly, GF has programming language support for linguistic constructs, and the community has a long experience of using it for many kinds of languages. Using GF, rather than a general purpose programming language, for grammar implementation is analogous to using YACC-like tools for programming language implementation: even though general purpose languages by definition are theoretically sufficient for implementing grammars, doing this would require many times more effort and low-level repetitive coding.

Thirdly, GF has APIs that allow it to be embedded in general purpose languages. Thus programmers can use GF functionalities — in particular, the RGL — without writing any GF code at all. Reproducing the knowledge contained in the RGL, rather than importing it via embedded grammars, would involve person decades of duplicate work.

This said, Abstract Wikipedia is a challenge that exceeds previous applications of GF, or any other NLG project, by at least two orders of magnitude: it involves almost ten times more languages and at least ten times more variation in content than any earlier project.

## Acknowledgements

I am grateful to Denny Vrandečić for inspiring discussions about the Abstract Wikipedia project and to the anonymous referees for insightful comments on this paper. Special thanks go to Roussanka Loukanova for her extraordinarily supportive editorial help.

## References

1. Angelov, K., Bringert, B., Ranta, A.: Speech-enabled hybrid multilingual translation for mobile devices. In: Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics, pp. 41–44. Gothenburg, Sweden (2014)
2. Angelov, K., Camilleri, J., Schneider, G.: A framework for conflict analysis of normative texts written in controlled natural language. The Journal of Logic and Algebraic Programming **82**, 216–240 (2013)
3. Angelov, K., Lobanov, G.: Predicting Translation Equivalents in Linked WordNets. In: The 26th International Conference on Computational Linguistics (COLING 2016), p. 26 (2016)
4. Angelov, K., Ranta, A.: Implementing Controlled Languages in GF. In: CNL-2009, Controlled Natural Language Workshop, Marettimo, Sicily, 2009 (2009)
5. Bamutura, D., Ljunglöf, P., Nabende, P.: Towards Computational Resource Grammars for Runyankore and Rukiga. In: Language Resources and Evaluation (LREC) 2020, pp. 2846–2854. Marseille, France (2020)
6. Bender, E.M., Koller, A.: Climbing towards NLU: On meaning, form, and understanding in the age of data. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 5185–5198. Association for Computational Linguistics, Online (2020). DOI 10.18653/v1/2020.acl-main.463. URL https://aclanthology.org/2020.acl-main.463
7. Burke, D.A., Johannisson, K.: Translating Formal Software Specifications to Natural Language / A Grammar-Based Approach. In: P. Blache and E. Stabler and J. Busquets and R. Moot (ed.) Logical Aspects of Computational Linguistics (LACL 2005), *LNCS/LNAI*, vol. 3492, pp. 51–66. Springer (2005). http://www.springerlink.com/content/?k=LNCS+3492
8. Camilleri, J.J.: A Computational Grammar and Lexicon for Maltese. Master's thesis, Chalmers University of Technology, Gothenburg, Sweden (2013). URL http://academic.johnjcamilleri.com/papers/msc2013.pdf
9. Dada, A.: Implementation of the Arabic Numerals and Their Syntax in GF. In: Proceedings of the 2007 Workshop on Computational Approaches to Semitic Languages: Common Issues and Resources, Semitic '07, p. 9–16. Association for Computational Linguistics (2007)
10. Dada, A.E., Ranta, A.: Implementing an Open Source Arabic Resource Grammar in GF. In: 20th Arabic Linguistics Symposium. Western Michigan University March 3-5 2006 (2006)
11. Dannélls, D., Damova, M., Enache, R., Chechev, M.: Multilingual online generation from semantic web ontologies. In: Proceedings of the 21st international conference on World Wide Web, pp. 239–242. ACM (2012)
12. Détrez, G., Ranta, A.: Smart paradigms and the predictability and complexity of inflectional morphology. In: EACL 2012 (2012)

13. Diriye, O., Folkesson, F., Nilsson, E., Nilsson, F., Nilsson, W., Osolian, D.: Multilingual Text Robots for Abstract Wikipedia. Bachelor's thesis, Chalmers University of Technology, Gothenburg, Sweden (2022)
14. Dymetman, M., Lux, V., Ranta, A.: XML and multilingual document authoring: Convergent trends. In: Proc. Computational Linguistics COLING, Saarbrücken, Germany, pp. 243–249 (2000)
15. Fellbaum, C.: WordNet: An Electronic Lexical Database. MIT Press (1998)
16. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In: C. Baroglio, P.A. Bonatti, J. Małuszyński, M. Marchiori, A. Polleres, S. Schaffert (eds.) Reasoning Web, Fourth International Summer School 2008, no. 5224 in LNCS, pp. 104–124. Springer (2008)
17. Grau Francitorra, P.: The Linguistic Structure of Wikipedia. Master's thesis, University of Gothenburg, Gothenburg, Sweden (2022)
18. Hallgren, T., Ranta, A.: An Extensible Proof Text Editor. In: M. Parigot, A. Voronkov (eds.) LPAR-2000, *LNCS/LNAI*, vol. 1955, pp. 70–84. Springer (2000). http://www.cse.chalmers.se/~aarne/articles/lpar2000.pdf
19. Hammarström, H., Ranta, A.: Cardinal Numerals Revisited in GF. In: Workshop on Numerals in the World's Languages, Dept. of Linguistics, Max Planck Institute for Evolutionary Anthropology, Leipzig (2004)
20. Kaljurand, K., Kuhn, T.: A multilingual semantic wiki based on Attempto Controlled English and Grammatical Framework. In: The Semantic Web: Semantics and Big Data, pp. 427–441. Springer (2013)
21. Khegai, J., Nordström, B., Ranta, A.: Multilingual Syntax Editing in GF. In: A. Gelbukh (ed.) Intelligent Text Processing and Computational Linguistics (CICLing-2003), Mexico City, February 2003, *LNCS*, vol. 2588, pp. 453–464. Springer-Verlag (2003)
22. Kituku, B., Nganga, W., Muchemi, L.: Leveraging on Cross Linguistic Similarities to Reduce Grammar Development Effort for the Under-Resourced Languages: a Case of Kenyan Bantu Languages. In: 2021 International Conference on Information and Communication Technology for Development for Africa (ICT4DA), pp. 83–88 (2021). DOI 10.1109/ICT4DA53266.2021.9672222
23. Kuhn, T.: A Survey and Classification of Controlled Natural Languages. Computational Linguistics **40**(1), 121–170 (2014)
24. Le, J., Zhou, R.: Multilingual Language Models for the Evaluation and Selection of Auto-Generated Abstract Wikipedia Articles. Master's thesis, Chalmers University of Technology, Gothenburg, Sweden (2022)
25. Listenmaa, I., Kaljurand, K.: Computational Estonian Grammar in Grammatical Framework. In: 9th SaLTMiL Workshop on Free/open-Source Language Resources for the Machine Translation of Less-Resourced Languages, LREC 2014, Reykjavík (2014)
26. Marais, L., Pretorius, L.: Exploiting a multilingual semantic machine translation architecture for knowledge representation of patient information for covid-19. pp. 264–279 (2021)
27. Montague, R.: Formal Philosophy. Yale University Press, New Haven (1974). Collected papers edited by Richmond Thomason
28. Nivre, J., de Marneffe, M.C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C.D., McDonald, R., Petrov, S., Pyysalo, S.m., Silveira, N., Tsarfaty, R., Zeman, D.: Universal Dependencies v1: A Multilingual Treebank Collection. In: Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016), pp. 1659–1666. European Language Resources Association (ELRA), Portorož, Slovenia (2016). URL https://www.aclweb.org/anthology/L16-1262
29. Power, R., Scott, D.: Multilingual authoring using feedback texts. In: COLING-ACL (1998)
30. Pretorius, L., Marais, L., Berg, A.: A GF miniature resource grammar for Tswana: modelling the proper verb. Language Resources and Evaluation **51**(1), 159–189 (2017)

31. Ranta, A.: Modular Grammar Engineering in GF. Research on Language and Computation **5**, 133–158 (2007)
32. Ranta, A.: Grammars as Software Libraries. In: Y. Bertot, G. Huet, J.J. Lévy, G. Plotkin (eds.) From Semantics to Computer Science. Essays in Honour of Gilles Kahn, pp. 281–308. Cambridge University Press (2009)
33. Ranta, A.: The GF Resource Grammar Library. Linguistics in Language Technology **2** (2009)
34. Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford (2011)
35. Ranta, A.: Translating between language and logic: What is easy and what is difficult. In: N. Bjørner, V. Sofronie-Stokkermans (eds.) Automated Deduction – CADE-23, pp. 5–25. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
36. Ranta, A., Angelov, K., Hallgren, T.: Tools for multilingual grammar-based translation on the web. In: Proceedings of the ACL 2010 System Demonstrations, pp. 66–71. Uppsala, Sweden (2010). URL https://aclanthology.org/P10-4012.pdf
37. Ranta, A., Détrez, G., Enache, R.: Controlled Language for Everyday Use: the MOLTO Phrasebook. In: CNL 2012: Controlled Natural Language, *LNCS/LNAI*, vol. 7175 (2010)
38. Ranta, A., Tian, Y., Qiao, H.: Chinese in the Grammatical Framework: Grammar, Translation, and Other Applications. In: Proceedings of the Eighth SIGHAN Workshop on Chinese Language Processing, ACL, pp. 100–109. Beijing, China (2015). URL http://www.aclweb.org/anthology/W15-3117
39. Reiter, E., Dale, R.: Building Natural Language Generation Systems. Cambridge University Press (2000)
40. Saludes, J., Xambo, S.: The GF mathematics library. In: THedu'11 (2011)
41. Stribrand, D.: Semantic Verification of Multilingual Documents. Master's thesis, Chalmers University of Technology, Gothenburg, Sweden (2022)
42. Van Eijck, J., Unger, C.: Computational semantics with functional programming. Cambridge University Press (2010)
43. Virk, S.: Computational linguistics resources for Indo-Iranian languages. Ph.D. thesis, Dept. of Computer Science and Engineering, Chalmers University of Technology and Gothenburg University (2013)
44. Vrandečić, D.: Building a Multilingual Wikipedia. Communications of the ACM **64**(4), 38–41 (2021). https://cacm.acm.org/magazines/2021/4/251343-building-a-multilingual-wikipedia/fulltext
45. Vrandečić, D., Krötzsch, M.: Wikidata: A free collaborative knowledgebase. Commun. ACM **57**(10), 78–85 (2014). DOI 10.1145/2629489. URL https://doi.org/10.1145/2629489
46. WikipediaContributors: Lsjbot. https://en.wikipedia.org/wiki/Lsjbot (2022). Online; accessed 20 April 2022
47. WikipediaContributors: Wikipedia:Content translation tool. https://en.wikipedia.org/wiki/Wikipedia:Content_translation_tool (2022). Online; accessed 20 April 2022
48. Zimina, E.: Fitting a Round Peg in a Square Hole: Japanese Resource Grammar in GF. In: JapTAL, vol. 7164, pp. 156–167. LNCS/LNAI, Kanazawa (2012)