Modular Approach to Automating Morphological Components in Grammar Engineering

Ekaterina Voloshina, Krasimir Angelov
Chalmers University of Technology, University of Gothenburg
Gothenburg, Sweden
ekaterina.voloshina@chalmers.se, krasimir.angelov@cse.gu.se

Abstract

Creating formal grammars is a time-consuming and complex task. We present a method to automatically create the morphological components of a formal grammar in Grammatical Framework. Our method is linguistically interpretable and modular, consisting of three stages: paradigm construction, extraction of inflectional classes, and prediction of inflectional classes. The modular structure allows human interventions after each stage. Moreover, our method supports encoding pre-existing language knowledge in form of Python APIs. Experiments show that automatically extracted morphological rules yield results comparable with manual grammars and that incorporating prior linguistic knowledge leads to improvement in low-resourced scenarios. Our findings show that our method simplifies the process of grammar development while preserving quality and interpretability.

Keywords: grammar engineering, computational morphology, under-resourced languages

1. Introduction

Our overall aim is the formalization of the syntax and the morphology of the world's languages in the form of a grammatical library that can be used for applications. Moreover, the formalization must be human readable and verifiable.

This is an overly ambitious goal, but it can be related to a number of other initiatives. The Universal Dependencies (UD) project (De Marneffe et al., 2021) collects treebanks for a multitude of languages annotated with a compatible schema. Wiktionary contains morphology and translation relations between hundreds of languages. Grammatical Framework (GF) (Ranta, 2011) provides a library of formal grammars for more than 40 languages. GF WordNet (Angelov, 2020) is a semi-automatically created resource with WordNet based lexicons for 264 languages, which is moreover compatible with the GF library.

However, none of these resources fully realizes the goal. UD is a collection of examples, but the syntactic and morphological rules are not explicitly formalized. Wiktionary is a great source of information, but since it is structured as a wiki, the extraction of data for applications is a messy process. The GF library has been gradually developed by a community of people, but adding new languages requires native language expertise and is a slow process. In GF WordNet, for most of its 264 languages, only the lemmas are available. Only for 45 languages the lemmas are complemented with morphology and syntax.

With this work, our aim is to bring GF closer to Wiktionary by learning the morphological rules from examples, thus speeding up the grammar development and expanding the set of languages.

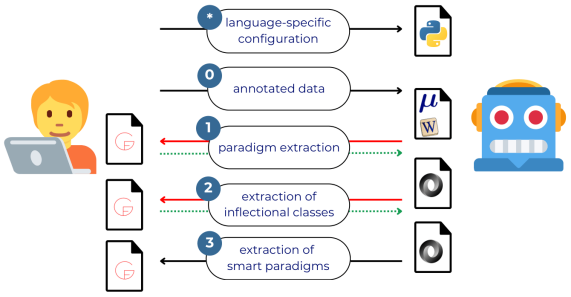

Figure 1: The pipeline of modules we use and their respective outputs. A user (on the left) provides annotated data and optionally configures parameters for the system, such as parameter values, default forms, etc., in forms of plugins written in Python. For each step, the algorithm produces the output converted to GF code (on the left) that the user can edit and feed to the system for the next step (green dashed lines).

The time required for building grammars is a bottleneck (Beemer et al., 2020a), but learning those fully automatically is a problem, if we want to guarantee correctness and human readability. In order to address that our system is organized into modules, where after each one a user can correct the output of the module. Moreover, the modules are independent of each other and different models can be integrated for specific subtasks in the future.

Since the primary requirement for the system is interpretability, we design a system with the paradigms as a central element to reflect the find-

Proceedings of the Fifteenth Language Resources and Evaluation Conference (LREC 2026), pages 11479–11494
11–16 May 2026.

ings of theoretical research on inflectional morphology. Several works (Matthews, 1972; Zwicky, 1985; Stump, 2001) suggest that paradigms are a central part of inflectional morphology. These claims are supported by research in psycholinguistics (Janssen and Penke, 2002; Sonnenstuhl et al., 2000). As a result of such a view of morphology, most of the grammar textbooks describe inflectional morphology in terms of paradigms.

Our experiments demonstrate in what aspects the automatic rule induction performs better than the manually defined rules and in which scenarios it is beneficial to utilize both manually written rules and automatic rule induction. Although previous work has addressed morphological inflection, class induction, or rule learning separately, to the best of our knowledge, this is the first system that integrates these tasks into a modular, interpretable pipeline that supports automatic code generation for grammar engineering.

Contributions (i) We propose a modular method for modeling morphology, summarized in Figure 1 which supports human intervention at every stage. (ii) We show how automatic extraction of morphological rules yields better results than writing them from scratch. (iii) We show how leveraging existing knowledge can improve results in extremely low-resourced scenarios. (iv) As a proof of concept, we add 7 new languages to GF library.[1]

2. Grammatical Framework

Grammatical Framework (GF) (Ranta, 2011) is a programming language specialized for the description of natural languages. GF is already supported with a library of grammars (Ranta, 2009) for more than 40 languages and a WordNet-like lexicon (Angelov, 2020, 2025) for 264 languages. Our code generator is specifically tailored to produce code compatible with these resources. We envision that in the future we will also generate syntactic rules by using UD treebanks or similar.

This section provides an overview of Grammatical Framework. One of the key design principles of the language is the abstract syntax – a collection of functions which provides a language independent API for the construction of natural language phrases. On the lexicon level, this means that every word is represented as a function with zero arguments. When this function is implemented separately for each concrete language, it usually realizes a word by constructing an inflection table. Similarly, syntactic functions take one or more arguments and combine them to construct a higher-level phrase. For example, adjectival modification is realized with the expression:

```
AdjCN (PositA nice_A) (UseN clothing_N)
```

Here the words are `nice_A` and `clothing_N`. The function `PositA` selects the positive (non-comparative) form of the adjective and builds a simple adjectival phrase; `UseN` lifts the noun to a common noun; finally `AdjCN` combines the two sub-phrases, while taking care of gender and number agreement. Given that these functions are implemented for several languages, the same expression can generate text in all of them, e.g. *nice clothing*, *snygg klädsel*, etc.

The concept of abstract syntax is inspired by the linguistic idea of interlingua, but except in limited domains, it is not a true interlingua. The reason is that different languages tend to use the same linguistic constructions for different purposes. The design is still useful and is more akin to how UD uses the same schema for diverse languages.

Since the morphology is language-specific, it is not part of the abstract syntax. On the abstract level, we only have the types `N`, `A`, `V`, etc., which are then defined different for each language by using parameters and other primitive types. For example, the type for nouns in Scandinavian languages is typically defined as:

```
param Species = Indefinite | Definite
param Case = Nom | Gen
param Number = Sg | Pl
param Gender = Utr | Neutr
lincat N = {s : Species => Case => Number => Str ;
            g : Gender
           }
```

Here `N` is a record, which contains the inflection table for each combination of the parameters (also called features) `Species`, `Case` and `Number`. Separately the field `g` stores the gender of the noun. See also Figure 2.

The above example is simpler, since, as it happens, the language has distinct forms for all combinations of features. This is not always the case, but can be described by using algebraic types. For example, European languages often have three adjective forms in singular – one form for every gender, and a common one for plural used for all genders. This corresponds to the parameter:

```
param GenNum = GSg Gender | GPl
```

where `GSg` takes an additional argument for gender, while the plural value `GPl` has no arguments (see Appendix A for more detailed description).

So far we have only covered how the paradigms are described. The actual inflections are implemented as a set of functions. Given a number of principal forms of the word, they return a value of the corresponding type (Détrez and Ranta, 2012).

3. Related Work

Computational morphology The field of computational morphology focuses on different approaches to modeling the morphological structures of languages, especially in under-resourced settings. Wiemerslage et al. (2022) provide a survey

[1]The code is available: https://github.com/GrammaticalFramework/rgl-learner/

of different morphological tasks, mentioning morphological (re)inflection, tagging, and segmentation, and paradigm clustering and completion. In this work, we focus on the latter task.

Most approaches to this problem are based on solutions for the morphological inflection problem, where models are trained to predict a form based on input *lemma + grammatical features*, therefore each form is predicted separately (Silfverberg et al., 2017; Silfverberg and Hulden, 2018; Kann et al., 2017). However, there are approaches designed specifically for the paradigm completion (PC) task. Cotterell et al. (2017b) suggest an algorithm based on neural graphical models for a paradigm completion task. The authors used the idea of principal parts to decode the paradigm in a joint way. Another approach by Kann et al. (2017) introduces a solution for the PC task by using new methods, such as paradigm transduction and source selection with high precision. The latter method chooses principal parts of the paradigm based on edit trees from lemmas to forms, making it more applicable in minimal-resource scenarios. However, the core models used in this work are also seq2seq models reused for the paradigm complection task.

Tools to support manual creation of morphological resources As for assisting linguists in creating morphological resources, Beemer et al. (2020b) used additional tools including a non-neural model made for the task of morphological inflection. However, most of the work so far has been done manually. Another work by Howell and Bender (2022) presents the methodology to build grammars, including morphological components, from language specifications and corpus data. In this paper, similar to the former work, we account for linguists to be able to control the output at the stage of generation, but similar to the latter work, we create linguistically plausible resources.

4. Definitions

For further understanding, we need to define paradigm, inflectional class, and inflection table (see Figure 2). We define a morphological feature F by the set of all possible values it can take: $F_i = \{f_1, ..., f_n\}$, where n is a number of feature values. For example, the morphological feature number can be defined in some language as: $\mathrm{Number} = \{\mathrm{Sg}, \mathrm{Dual}, \mathrm{Pl}\}$. Most features are atomic, but they can also be constructed by applying a constructor to another value. For example, the set of all values for GenNum is $\{\mathrm{GSg\ Utr}, \mathrm{GSg\ Neutr}, \mathrm{GPl}\}$.

If $F_1, ..., F_n$ are possible morphological features for a given part of speech, then normally a **paradigm** P is a Cartesian product:

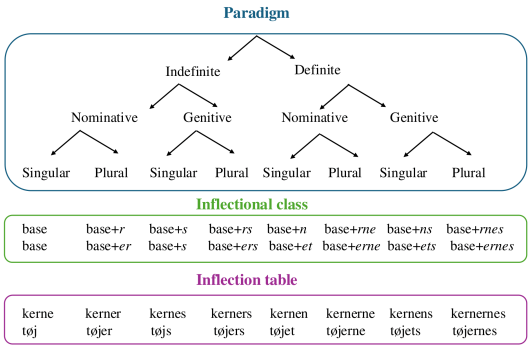

Figure 2: Example paradigm for a noun class in Danish. The top (in blue) shows the abstract paradigm structure, the middle row shows examples of noun inflectional classes defined by form patterns (in green), the bottom row shows examples of inflection tables for two nouns (in purple) that belong to the two inflectional classes above

$$P = F_1 \times ... \times F_n$$

Since in the Cartesian product the feature order is fixed, the paradigm can also be seen as a tree (Figure 2). The order comes from the order of the features in the data.

An **inflection table** IT, on the other hand, is a set of pairs:

$$IT = \{(w_i, f_i) | f_i \in P\},$$

where w_i is an inflection form and f_i is a morphosyntactic description.

A generalised version of an inflection table is an **inflectional class** IC where all members of a class share the same set of form patterns. The form patterns are generalized representations of all forms that have the same set of morphological features and share the same affixes. An example inflectional class for English nouns would be:

$IC : base \rightarrow \{(base, Sg), (base + s, Pl)\}$, where *base* is a variable that encodes a stem.

The essential part of our approach is building **smart paradigms** (Détrez and Ranta, 2012) – given some forms p_i from a paradigm, a smart paradigm SP can correctly reconstruct full paradigms:

$$SP : (p_1, ..., p_n) \rightarrow \{q_i, ..., q_k\}, \quad 1 \leq n < k,$$

where q_i is a cell of the inflection table.

This idea is closely related to the concept of principal parts in theoretical morphology (Finkel and Stump, 2007). Principal parts are the minimal subset of a paradigm from which it is possible to deduce all other forms in a paradigm. The classic example comes from Latin: all verb forms cannot be retrieved from one infinitive form, and therefore several forms are needed to define a full paradigm correctly.

5. Method

Our algorithm consists of three modules: paradigm construction, alignment of inflectional classes, and creation of smart paradigms (see Figure 1). Each step is automatic: running the pipeline produces a functional prototype of a grammar resource.

Moreover, after each step, the algorithm produces a readable programming code in GF, which can be manually inspected and edited. During compilation, GF can also be instructed to produce a JSON representation of the grammar, which our algorithm uses as input for the next module. Therefore, the module produced could be easily edited and fed to the next step in the form of JSON.

5.1. Module 0: Data preparation

Currently, we support two sources: UniMorph (Sylak-Glassman, 2016) and Wiktionary. UniMorph is a standardized format for morphological tasks with limited but better curated data. At the same time, Wiktionary usually contains more data and more languages, which we access through Wiktextract (Ylonen, 2022).

The system is modular and makes it possible to add other sources in the future, as the data access is handled through source-specific plugins. The plugins are simply Python modules, which the framework detects and then uses them to call specific functions for different tasks. This lets the user to alter the default behavior depending on data source and language. For instance, the role of the source plugin is to transform the input format to a list of word forms and morphological tags. Similarly, with language-specific plugins, the user can give instructions for how to clean up the input data (see Appendix B for more detailed description of plugins).

5.2. Module 1: Paradigm extraction

We extract one paradigm per part of speech and lexicon (a list of inflection tables). Recall that a paradigm is a Cartesian product of morphological features. Therefore, the first step is to define the morphological features $\{F_i\}$ and their values $\{f_j\}$ for a given language and to map them to GF features.

The mapping from source tags to morphological features following the naming convention adopted by the authors of the existing GF grammars is defined in the source-specific plugins. Note that not all tags are included. For example, Wiktionary uses tags to indicate archaic forms, transliterations, or regional variations. These are not kept in the grammar.

Another issue is that tags are sometimes used inconsistently with the general guidelines. In such cases, in a language-specific plugin, a user can define a correct mapping for the particular language.

When the algorithm has collected all morphological features, it defines the order based on the average position of a tag in the source. The algorithm also keeps track of all possible combinations of the features.

Based on the defined order and possible combinations of morphological features, the algorithm produces a paradigm tree (Figure 2). This is basically a trie, where the edges are labeled with features. As the last step, the algorithm constructs an inflection table for each lemma by filling the edges of the trie with the corresponding forms. Some edges are left empty if the combination of the features is possible, but no form is found for the lemma.

In general, the main goal of this module is to automatically clean up the data and organize them in a convenient way that leads to correct GF code.

The user's input is still needed since the source annotations may be incomplete and inconsistent. For example, for some nouns Wiktionary may list only singular and plural forms, while, for others, case inflected forms are also provided. In situations like this, forms marked only as singular/plural are usually to be understood as forms in nominative. The general algorithm does not make such automatic assumptions, but instead the user may write a language-plugin function, which manipulates the final form of the tree.

The user may also decide to alter the paradigm tree, simply in order to make the development of the future syntactic functions easier.

At the end of this phase, the generated code contains a set of parameters similar to the ones in Section 2, and a lexicon where the entries look like:

```
lin tøj_N = mkNoun "tøj" "tøjer" "tøjs"
            "tøjers" "tøjet" "tøjerne"
            "tøjets"  "tøjernes"
```

here `mkNoun` is an automatically generated function which fills in a paradigm with the given forms (see Appendix C for full examples of the output of each modules).

5.3. Module 2: Extraction of inflectional classes

This module is based on the idea of the longest common subsequence (LCS) adapted for paradigm extraction from Ahlberg et al. (2015). The first step is to extract LCS for each inflection table ($base$), which ideally corresponds to a word's *stem*. Note that the base can be discontinuous and then represented as $base_1, base_2$, etc. Other parts of the words can be seen as *affixes*. Recall the example from Figure 2. For the word *kerne*, the LCS is

'kerne' and this is taken as the base, so the plural form *kerner* would be parsed as the pattern $base + r$.

When all forms in each inflection table are represented as patterns of bases and affixes, the next step is to merge into one class all inflection tables that share the same patterns. If the data is sparse, and not all forms are available, some paradigms might not be merged. For example, for most English nouns, a form pattern for plural would look like *base+s*. If for some words, such as *scissors*, only plural forms are attested in the data, the plural pattern would be simply *base*, which will result in an additional inflectional class.

To solve this problem, we first merge classes into one, if inflectional class A includes more affixes than class B and forms from B can be parsed as from class A. Recall the example above: we examine that word *scissors* with pattern *base* (class B) could also be parsed as pattern *base+s* (class A). If so, we eliminate class B and merge all the forms into class A.

Some inflectional classes contain missing values but it is not possible to merge them unambiguously with another class. In this case, we take all classes that overlap in some forms and patch a missing form from these classes by majority voting. We also explicitly mark in the generated formal grammar that this form pattern has been guessed and not attested in the data.

To support prediction of all forms in the inflectional class, the function should be able to parse a lemma to extract all bases. In some cases, words are parsed using several bases, for example, languages with transfixes, such as Arabic, Hebrew (Semitic, Afroasiatic), or Germanic words where the stem vowel changes. It is especially difficult to parse if the bases are contiguous in a lemma and the infixes are injected later. For example the Albanian *kalë* becomes *kuaj* in plural. This means that we have $base_1 + base_2 +$ **lë** and $base_1 +$ **u** $+ base_2 +$ **j** for singular and plural.

In order to figure out how to unambiguously extract the bases from a lemma, we need to define an unambiguous regular expression for parsing. We inspect all known inflection tables belonging to the same class. Typically, all bases, except one, have a constant length across classes. This is used as a constraint in the case when the bases are contiguous. In rare cases, it turns out that more than one base is of various lengths. An example are digraphs like **rr**, **dh**, etc. in Albanian, which should really be counted as one phoneme but are written as two characters. This is resolved by choosing the base with the smallest number of possible values and explicitly matching the pattern on the possible strings.

There is also an optional step of generalization

Algorithm 1 Learning smart paradigms

```
while more principal parts possible do
    i ← 1
    do
        H_i ← predictClass(trainData, form, i)
        i ← i + 1
    while (ϵ < H_i < H_{i−1}) and (i ≤ depth)
    errors ← evaluate(evalData)
    form ← chooseForm(errors)
end while
```

of the paradigm: if there are multiple bases, the algorithm can produce more inflectional classes than necessary. For example, a common scenario in Greek is the stress movement: the algorithm initially extracts the base, such as $base_1$+ω+$base_2$, where ω is singled out only because it appears as stressed in another form. Therefore, the algorithm creates different classes for different stem vowels. To avoid this problem, one can use two forms (one with stressed and one with non-stressed stems) to extract patterns: $base_1$+ω+$base_2$ becomes $base_1$, and $base_1$+ώ+$base_2$ becomes $base_2$. However, this is not the default option because it always requires at least two forms, while for future usage we want to have the option to predict inflectional class with only one form.

The final result is a file with all found inflectional classes represented as functions. Each function takes a lemma as an argument and produces the table with all forms. We also update the dictionary where now only the citation forms are listed, while the rest are generated by applying the functions:

```
lin tøj_N = mkN021 "tøj"
```

here the number in the name `mkN021` is an incremental number generated to identify the class. We no longer need to list all forms, but on the other hand we need to know the inflectional class.

5.4. Module 3: Creation of smart paradigms

The goal of this module is to infer rules to predict an inflectional class of unknown words in the grammar. In principle, any model can be used, as long as it allows to extract the pattern-matching rules as readable code. While transformer-based models currently achieve the state-of-the-art results on many morphological tasks, the lack of transparency and interpretability makes them unsuitable for our pipeline. Therefore, we adopt a decision-tree based implementation (see Algorithm 1), formulating the problem as a classification task, where the classes are all extracted inflectional classes for a given part-of-speech.

The first step is to define whether the language uses primarily, prefixes or suffixes (Dryer, 2013).

Although this information could be taken from external resources, we prefer not to rely on that because of their scarcity. Similarly to Cotterell et al. (2017a), we count how many form patterns start with an affix (rather than a stem) and how many form patterns end with an affix

The algorithm starts with a lemma form and looks at affixes of all lemmas in the training data of length 1. Then for each possible affix a of length 1, it assigns the most likely inflectional class based on the training data:

$$c_i = \text{argmax}_c\, p(c|a_i(1)),$$

where $a_i(l)$ is an affix of length l and $p(c|a_i(1))$ is the probability that a word belongs to the class c given the affix a_i.

Then for each affix, the algorithm computes the entropy $H_{i,1}$. If extending the affix reduces entropy, i.e $H_{i,2} < H_{i,1}$, then the algorithm produces one more rule. The process continues until maximum length is reached or the entropy stops decreasing with each $l + 1$.

After one iteration, the algorithm chooses a new form with the highest error rate. We assume that the algorithm would make more mistakes in one of the principal parts as they are unpredictable from a lemma by definition. Then, the algorithm proceeds to the next stage where, this time, still ambiguous classes are discriminated by considering the new form, i.e. the goal is to maximize likelihood:

$$c_i = \text{argmax}_c\, p(c|a_i(l_i), a_k(l_k)),$$

, where $a_i(l_i)$ is the affix of length l_i of a lemma, and $a_k(l_k))$ is the affix of length l_k of a chosen principal part. The process ends when the maximum number of principal parts is reached.

This design allows the model to be initialised with existing morphological rules (for example, from a grammar book) and learn rules that are not covered yet, as well as define what principal parts to use.

Once we have learned the smart paradigms, we can simply define a lexical entry as:

```
lin tøj_N = mkN "tøj"
```

for regular words. If the word is irregular, we need to provide several arguments (principal parts):

```
lin tøj_N = mkN "tøj" "tøjer"
```

This is in line with the design in (Détrez and Ranta, 2012).

6. Experimental Setup

We test our module in three different ways: its performance in comparison to manually defined rules and its performance on different amounts of data and with different types of human intervention.

Comparison to manually defined rules As our goal is to assist in the generation of morphological resources, we compare the generated grammar components with manually written grammars. To compare the coverage of detected rules, we choose 5 languages from different language groups that are already supported in the GF library Ranta (2009): Modern Greek (IE), French (Romance, IE), Slovak (Slavic, IE), Hungarian (Finno-Ugric), and Finnish (Finno-Ugric).

One limitation when working with manually written grammars is the variety of ways in which the rules can be written. To make the comparison fair, we choose grammars that use pattern matching in one or more forms. We run the system in three ways:

- **M2+M3**: the set of inflectional classes is defined by the inflectional classes described in the manual grammar, and the rules are based on the manual grammars. We rewrite the rules as input to our algorithm as we specify them in a language-specific plugin.
- **M2+A3**: the set of inflectional classes is the same as in the manual grammar but the system automatically learns the rules.
- **A2+A3**: we train the algorithm on the entire training data, so both inflectional classes and rules are automatically induced.

For Greek we generalize the forms, so each inflectional class takes at least two forms.

We use UniMorph datasets for all languages except Greek. We use 80% of the data set as training data and the rest as test data.

Effect of training size We run the algorithm in each setting with several different thresholds of the training data (each repeated 5 times): 50, 100, 400, 700 and 1000 inflection tables (i.e. all word forms for *n* lemmas), to show how much data is required for performance to stabilize. The languages included in this experiment are the new languages that we added to the GF-RGL library: Albanian (IE), Armenian (IE), Faroese (Germanic, IE), Irish (Celtic, IE), Macedonian (Slavic, IE), Kazakh (Turkic), and Scottish Gaelic (Celtic, IE). The choice of the languages is dictated by the interest in the community.

Human-assisted performance This experiment is meant to test the limits of the algorithm and to show how low-cost manual intervention can improve the performance.

We include five languages in our experiment: Armenian, Faroese, Irish, Kazakh, and Macedonian. In this experiment, we are testing whether providing a few rules that can be easily found in grammar books will improve the performance in the earlier stages when there is not enough data.

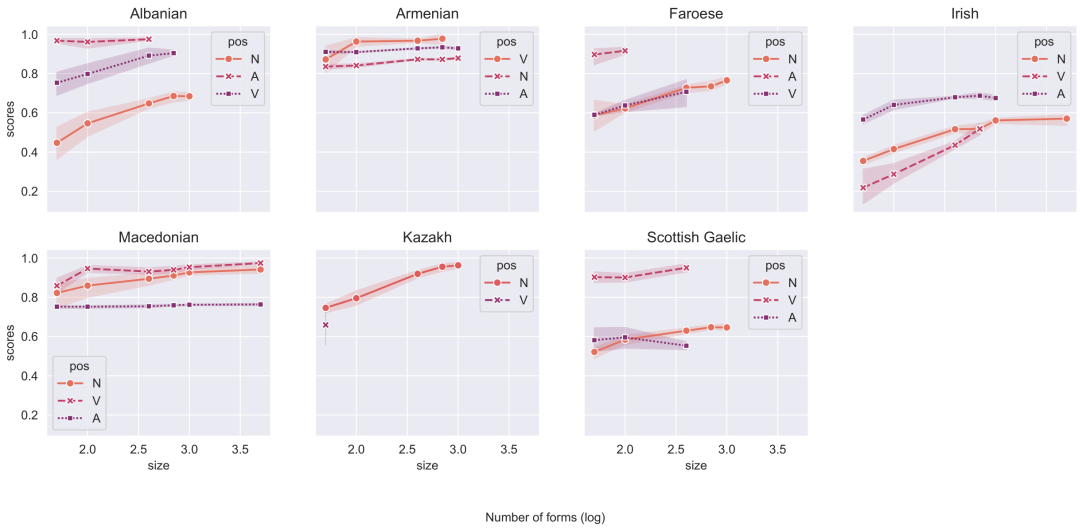

Figure 3: Effect of training data size on accuracy across languages. The X-axis shows the amount of data (in log-scale with base 10). The y-axis shows the accuracy-per-form, i.e. percentage of all forms produced correctly. The curves stabializes around the threshold of 400 inflection tables.

We run the algorithm in three different setups: with no corrections, with pre-defined principal parts, and with pre-defined rules. We consult the following grammar books to define principal parts and rules: Dum-Tragut (2009) (Armenian), Lockwood (1977) (Faroese), Doyle (2001) (Irish), Somfai Kara (2002) (Kazakh), Friedman (2002) (Macedonian).

7. Results

language	POS	M2+M3	M2+A3	A2+A3
Greek	N	0.734	0.659	0.886
Finnish	V	0.854	0.834	0.842
French	V	0.798	0.825	0.834
Hungarian	N	0.606	0.68	0.807
Slovak	N	0.5	0.787	0.941

Table 1: Accuracy-per-form (percentage of all forms produced correctly) with manually written rules and automatically generated rules. M2 includes only inflectional classes that are included in the manual grammar, M3 includes rules from the manual grammar. A2 are the automatically extracted inflectional classes and A3 are the automatically extracted rules

Comparison to manual grammars As Table 1 illustrates, the generated rules cover the morphological variation at least as well as well-tested manually written grammars (French and Finnish). However, we significantly improve on the newer grammars (Greek, Slovak, and Hungarian), which were developed in a shorter time and have not been extensively used. This shows that our system is useful as a support tool in grammar development.

Which inflectional classes are included in the grammar is important: although the set of inflectional classes based on manual grammars (M2) is smaller, in case of French and Finnish, the algorithm performs as well as on the full set of inflectional classes (M2+A3 shows almost the same result as A2+A3). In cases of Greek, Slovak and Hungarian, we see that training on a full set of inflectional classes gives significant gain in performance. It supports the hypothesis that the grammar description of classes is simplified and does not include all possible classes (Silfverberg et al., 2018a).

Comparison of manual (M3) and automatically extracted (A3) rules shows that for Hungarian, French, and Slovak datasets, the algorithm finds rules that perform better, while for Greek, manual rules lead to better performance. One possible explanation is that, in the manually written Greek grammar, the choice of the second form is not fixed: several different forms are used for different inflectional classes. While this leads to better performance, it is more convenient to use the grammar with determined forms.

Effect of training size Figure 3 shows the effect of the training size. Across the board, when given at least 400 inflection tables in the training data, the algorithm produces stable results. As a practical implication, it shows that a dataset of such size would be sufficient.

Effect of manual interventions We then compare the performance of the algorithm trained with

Language	POS	Rules	Size: 50			Size: 100			Size: 1000		
			NC	PP	GR	NC	PP	GR	NC	PP	GR
Armenian	N	11	**0.836**	0.823	**0.836**	0.841	0.849	**0.858**	0.879	**0.881**	**0.881**
	A	1	0.911	0.911	0.911	0.909	0.909	**0.915**	0.928	0.928	**0.929**
	V	5	0.872	0.872	0.872	0.963	0.963	0.963	NA	NA	NA
Faroese	N	7	0.587	0.594	**0.654**	0.623	0.656	**0.691**	0.766	0.797	**0.802**
	A	2	0.897	0.897	0.897	0.916	0.916	0.916	NA	NA	NA
	V	3	**0.590**	0.534	0.534	**0.638**	0.589	0.589	NA	NA	NA
Irish	N	3	0.355	**0.357**	0.346	0.416	**0.417**	0.410	0.561	0.561	0.561
	A	2	**0.566**	0.565	*0.562*	0.640	0.645	**0.670**	0.675	0.677	**0.683**
	V	1	*0.219*	*0.219*	**0.255**	0.288	0.288	**0.309**	NA	NA	NA
Kazakh	N	4	0.746	0.746	0.746	0.796	0.796	0.796	0.963	0.963	0.963
	V	2	*0.659*	*0.659*	*0.659*	NA	NA	NA	NA	NA	NA
Macedonian	N	10	0.822	0.834	**0.862**	0.86	0.881	**0.903**	0.927	0.956	**0.957**
	A	3	0.752	0.752	**0.756**	0.752	0.752	**0.756**	**0.762**	0.760	**0.762**
	V	12	**0.859**	0.838	0.838	0.947	**0.960**	**0.960**	0.953	**0.976**	**0.976**

Table 2: Accuracy-per-form (percentage of all forms produced correctly) for different user-involving scenarios across different training data sizes (50, 100 or 1000 inflection tables). Three possible scenarios include no input from a user (NC), pre-defined principal parts (PP), and with initialization with grammar rules (GR). NA signifies that the dataset did not contain enough data. The results are given by each part-of-speech separately and the number of rules is specified. All the results are averaged over 5 runs, results with standard deviation over 0.1 are given in italics

no input from a user, with user-defined principal parts, and with user-defined rules as part of the initialisation. Table 2 summarizes the results. The results are language-dependant: for example, for Kazakh no interventions improved the quality.

As grammars vary in the way they describe rules, we explicitly mention how many rules were used in initialization. Across the board, adding more than 5 rules shows improvement. However, even one rule improved the performance for verbs in Irish, as the dataset did not have enough examples.

Moreover, as an overall trend, the more data the algorithm gets, the smaller the difference between fully automatic and pre-initialized settings becomes.

8. Discussion

Using existing knowledge Our experiments show that including any existing knowledge, such as knowledge about principal parts or rules from grammar descriptions, can improve the quality of the algorithm, especially in low-resource scenarios. It suggests that combination of existing knowledge and data-driven approaches gives a promising direction in low-resource scenarios. One challenge in this situation is to make the integration as easy as possible for users. This is the reason why, unlike previous works where seq2seq approach was used, we split the problem into several modules, which means that the inflectional classes can be automatically induced. In that case, the user does not have to write the rules for the inflectional class from scratch but rather they can write the rules identifying such class.

Effect of number of inflectional classes It is a known fact that computational models of morphology extract more classes (Détrez and Ranta, 2012; Silfverberg et al., 2018a). However, our results show that if the inflection tables are properly chosen, the model can cover most of the data, and the gain from more paradigms is not big. It leads to the conclusion that even with a small amount of data, it can be possible to build a computational model, if the data is well-curated.

Implications for grammar engineering As our second experiment shows, around 400 inflection tables are enough to give stable results. If there is enough data, the involvement of the user would be mostly post-processing: the system warns about tags that are unknown to it, the user can check generated results after each step, correct them, or define some metavariables, such as order of the morphological features, to change the outlook of the generated grammar. However, if there are not enough data, the user can initialize the algorithm with existing knowledge from grammar books, as the third experiment shows.

9. Conclusion

We have presented a modular system, which automates the creation of morphological components

for formal grammars by combining paradigm modeling, induction of inflectional classes, and learning rules for inflectional class classification. To the best of our knowledge, this is the first system that automatically creates grammars and supports corrections and initialization based on prior knowledge.

Unlike manual creation, our approach is more scalable and less time-consuming. Moreover, our experiments show that in most cases our system performs better than the manually written rules. We argue that this is due to grammar books not mentioning all exceptions that are found in data.

Unlike neural approaches, our method is computationally cheaper and provides user flexibility for encoding prior knowledge, such as rules that identify different inflectional classes. Unlike previous seq2seq approaches, building the system around paradigms allows to minimize manual work and makes the output better aligned with theoretical representation of inflectional morphology.

As a practical contribution, we created morphological components for 7 new languages.

Limitations

The main limitation of our method is that it requires structured input with morphological annotation, although it is not limited to one specific formalism of annotation.

As for the output of our algorithm, we support code generation for the Grammatical Framework but one can adapt to any possible format.

Ethical considerations

Responsible use of language technology is especially important in minority or dying languages. In those cases, generation of erroneous output is dangerous since new language users might rely on the computer output and thus start learning the mistakes produced by the machine. The possibility to inspect and correct the computational model is therefore essential.

There are several sources of errors in our approach. Annotation errors in the original data may lead to errors in the predictions. Those cases can be solved by either fixing the annotations or preprocessing via the plugins mechanism.

When we merge paradigms, we introduce morphological patterns for forms that were not attested. These we mark in the generated source code and should be manually inspected.

When a lexicon for a particular application is built, new words can be introduced by applying the smart paradigm to the lemma. The paradigm will do its best to guess the rest of the forms, but for reliability a native speaker must inspect the output. When necessary, more than one form must be provided to the paradigm to be able to correctly predict the full paradigm.

10. Bibliographical References

Malin Ahlberg, Markus Forsberg, and Mans Hulden. 2015. Paradigm classification in supervised learning of morphology. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1024–1029, Denver, Colorado. Association for Computational Linguistics.

Krasimir Angelov. 2024. A scalable database for syntactic and lexical resources. In *CLIRAI 2024: Computational Linguistics, Information, Reasoning, and AI*.

Sarah Beemer, Zak Boston, April Bukoski, Daniel Chen, Princess Dickens, Andrew Gerlach, Torin Hopkins, Parth Anand Jawale, Chris Koski, Akanksha Malhotra, Piyush Mishra, Saliha Muradoglu, Lan Sang, Tyler Short, Sagarika Shreevastava, Elizabeth Spaulding, Testumichi Umada, Beilei Xiang, Changbing Yang, and Mans Hulden. 2020a. Linguist vs. machine: Rapid development of finite-state morphological grammars. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 162–170, Online. Association for Computational Linguistics.

Sarah Beemer, Zak Boston, April Bukoski, Daniel Chen, Princess Dickens, Andrew Gerlach, Torin Hopkins, Parth Anand Jawale, Chris Koski, Akanksha Malhotra, et al. 2020b. Linguist vs. machine: Rapid development of finite-state morphological grammars. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 162–170.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, Géraldine Walther, Ekaterina Vylomova, Patrick Xia, Manaal Faruqui, Sandra Kübler, David Yarowsky, Jason Eisner, and Mans Hulden. 2017a. CoNLL-SIGMORPHON 2017 shared task: Universal morphological reinflection in 52 languages. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 1–30, Vancouver. Association for Computational Linguistics.

Ryan Cotterell, John Sylak-Glassman, and Christo Kirov. 2017b. Neural graphical models over strings for principal parts morphological paradigm completion. In *Proceedings of the*

15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers, pages 759–765, Valencia, Spain. Association for Computational Linguistics.

Grégoire Détrez and Aarne Ranta. 2012. Smart paradigms and the predictability and complexity of inflectional morphology. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 645–653, Avignon, France. Association for Computational Linguistics.

A. Doyle. 2001. *Irish*. Languages of the world: Materials. LINCOM Europa.

Matthew S. Dryer. 2013. Prefixing vs. suffixing in inflectional morphology (v2020.4). In Matthew S. Dryer and Martin Haspelmath, editors, *The World Atlas of Language Structures Online*. Zenodo.

Jasmine Dum-Tragut. 2009. Armenian.

Raphael Finkel and Gregory Stump. 2007. Principal parts and morphological typology. *Morphology*, 17:39–75.

Victor A Friedman. 2002. *Macedonian*. Lincom Europa Munich.

Omer Goldman and Reut Tsarfaty. 2021. Minimal supervision for morphological inflection. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2078–2088, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Kristen Howell and Emily M Bender. 2022. Building analyses from syntactic inference in local languages: An HPSG grammar inference system. *Northern european journal of language technology*, 8.

Ulrike Janssen and Martina Penke. 2002. How are inflectional affixes organized in the mental lexicon?: Evidence from the investigation of agreement errors in agrammatic aphasics. *Brain and Language*, 81(1-3):180–191.

Katharina Kann, Ryan Cotterell, and Hinrich Schütze. 2017. Neural multi-source morphological reinflection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 514–524, Valencia, Spain. Association for Computational Linguistics.

Katharina Kann and Hinrich Schütze. 2018. Neural transductive learning and beyond: Morphological generation in the minimal-resource setting. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3254–3264, Brussels, Belgium. Association for Computational Linguistics.

Jordan Kodner, Sarah Payne, Salam Khalifa, and Zoey Liu. 2023. Morphological inflection: A reality check. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6082–6101, Toronto, Canada. Association for Computational Linguistics.

William Burley Lockwood. 1977. *An introduction to modern Faroese*. Nám.

Peter Hugoe Matthews. 1972. *Inflectional morphology: A theoretical study based on aspects of Latin verb conjugation*, volume 6. CUP Archive.

Aarne Ranta. 2011. *Grammatical framework: Programming with multilingual grammars*, volume 173. CSLI Publications, Center for the Study of Language and Information Stanford.

Miikka Silfverberg and Mans Hulden. 2018. An encoder-decoder approach to the paradigm cell filling problem. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2883–2889.

Miikka Silfverberg, Ling Liu, and Mans Hulden. 2018a. A computational model for the linguistic notion of morphological paradigm. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1615–1626, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Miikka Silfverberg, Ling Liu, and Mans Hulden. 2018b. A computational model for the linguistic notion of morphological paradigm. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1615–1626.

Miikka Silfverberg, Adam Wiemerslage, Ling Liu, and Lingshuang Jack Mao. 2017. Data augmentation for morphological reinflection. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 90–99, Vancouver. Association for Computational Linguistics.

Dávid Somfai Kara. 2002. Kazak (languages of the world/materials 417.). *München: LINCOM Europa*.

Ingrid Sonnenstuhl, Meike Hadler, Sonja Eisenbeiss, and Harald Clahsen. 2000. Morphological paradigms in the mental lexicon. In *9th International Morphology Meeting, Vienna*.

Gregory T Stump. 2001. *Inflectional morphology: A theory of paradigm structure*, volume 93. Cambridge University Press.

Linda Wiechetek, Flammie Pirinen, Mika Hämäläinen, and Chiara Argese. 2021. Rules ruling neural networks - neural vs. rule-based grammar checking for a low resource language. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, pages 1526–1535, Held Online. INCOMA Ltd.

Adam Wiemerslage, Miikka Silfverberg, Changbing Yang, Arya McCarthy, Garrett Nicolai, Eliana Colunga, and Katharina Kann. 2022. Morphological processing of low-resource languages: Where we are and what's next. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 988–1007, Dublin, Ireland. Association for Computational Linguistics.

Tatu Ylonen. 2022. Wiktextract: Wiktionary as machine-readable structured data. In *Proceedings of the 13th Conference on Language Resources and Evaluation*, pages 1317–1325.

Arnold M Zwicky. 1985. How to describe inflection. In *Annual Meeting of the Berkeley Linguistics Society*, pages 372–386.

11. Language Resource References

Krasimir Angelov. 2020. A parallel WordNet for English, Swedish and Bulgarian. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 3008–3015.

Krasimir Angelov. 2025. An abstract multilingual WordNet. In *Global WordNet Conference 2025*.

Marie-Catherine De Marneffe, Christopher D Manning, Joakim Nivre, and Daniel Zeman. 2021. Universal dependencies. *Computational linguistics*, 47(2):255–308.

Aarne Ranta. 2009. The GF resource grammar library. *Linguistic Issues in Language Technology*.

John Sylak-Glassman. 2016. The composition and use of the universal morphological feature schema (unimorph schema). *Johns Hopkins University*, page 6.

A. Grammatical Framework

The abstract syntax serves as interlingual representation that is linearized differently in each language. For example, the following abstract syntax expression :

```
AdjCN (PositA nice_A) (UseN clothing_N)
```

represents an adjectival modification, and it is linearized into *nice clothing* in English, *snygg klädsel* in Swedish, and *vêtements agréables* in French. The concrete linearizations of rules define how this expression is realized both morphologically and syntactically.

Lexical items, such as `nice_A` and `clothing_N`, are functions that do not take any arguments and are defined as following:

```
English: nice_A = mkA "nice" "nicer"
Swedish: nice_A = mkA "snygg"
French:  nice_A = mkA "agréable" "agréable"

English: clothing_N = mkN "clothing"
Swedish: clothing_N = mkN "klädsel"
French:  clothing_N =
              mkN "vêtement" "vêtements"
                  masculine
```

`mkA` and `mkN` are functions that implement smart paradigms, which, given some forms, recreate a full inflection table for the given adjective or noun. These are precisely the functions that our system learns automatically.

The resulting expressions have types A and N respectively. These types represent what morphological features (defined as parameters) each part-of-speech has in a particular language. For example, nouns in English are defined in the following way, as an English noun has four forms (*girl, girl's, girls, girls'*):

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

lincat N = {s : Number => Case => Str ;
            g : Gender} ;
```

Note that although English, is generally seen as a genderless language, the full grammar still has a gender parameter. It shows up in rare cases when reflexives are used, e.g. himself/herself/itself.

In Swedish, nouns additionally encode definiteness and inherent gender:

```
param Species = Indefinite | Definite
param Gender = Utr | Neutr
lincat N = {s : Species => Case =>
                 Number => Str ;
            g : Gender
           }
```

While smart paradigms define morphology, other functions in the abstract syntax represent syntactic composition. For example, `UseN` is a function that

creates a common noun phrase that consists of a single noun. This is usually an identify function, but there are also exceptions. For instance, in morphologically rich languages the plain noun may contain only stems, which are extended to full forms by `UseN`.

`PositA`, on the other hand, chooses the positive form of an adjective and produces an adjectival phrase.

Finally, adjectival modification is defined by `AdjCN`, which combines the adjective phrase with the common noun to form another common noun. To produce a complete noun phrase, one would apply further syntactic combinators which will add a determiner as well (See Ranta (2009)).

In English, the last rule mainly defines the position of an adjective:

```
AdjCN ap cn = {
   s = \\n,c => preOrPost ap.isPre
                  (ap.s ! agrgP3 n cn.g)
                  (cn.s ! n ! c) ;
   g = cn.g
   };
```

but it also ensures that there is a gender, number, and case agreement. The gender in particular is relevant only if the adjectival phrase contains reflexives, while the only non-trivial case is genitive.

In French, the gender and number agreement is more pronounced and is implemented in a very similar way:

```
AdjCN ap cn = {
   s = \\n => preOrPost ap.isPre
                (ap.s ! genNumPos2Aform cn.g n
                                        ap.isPre)
                (cn.s ! n) ;
   g = cn.g ;
  };
```

Finally, its linearization in Scandinavian languages, also includes agreement in number, definiteness, and gender:

```
AdjCN ap cn = {
   s = \\n,d,c =>
         preOrPost ap.isPre
          (ap.s ! agrAdj (gennum (ngen2gen cn.g) n) d)
          (cn.s ! n ! d ! c) ;
   g = cn.g ;
   isMod = True
  } ;
```

B. Plugins

Plugins are Python modules which make it possible to preprocess data and override automatic decisions. To start with, the input data are accessed via a source specific plugin, currently only two – Unimorph and Wiktionary. In addition, for each combination of language and source, we can write a different language-specific plugin.

Source Plugins The main component in the source plugins is the extract function which reads the input data from a custom format and returns a quadruple of lemma, part of speech, list of forms and lemma tags. Currently, the only possible lemma tag is the gender for nouns. The list of forms contains pairs of forms with the associated tags. When reading the raw data, the plugin also calls the language-specific preprocess function, which can perform arbitrary cleanup on the raw data. For instance, the Kazakh data in Wiktionary contained a mixture of Latin and Cyrillic letters that had to be cleaned up.

The other main ingredient is the mapping from source-specific tags to prefered GF names. For example, the following are the tags in Wiktionary for a grammatical person:

```
params = {
  'first-person':  ('P1','Person'),
  'second-person': ('P2','Person'),
  'third-person':  ('P3','Person')
}
```

while these are the corresponding tags in Unimorph:

```
params = {
  '1': ('P1', 'Person'),
  '2': ('P2', 'Person'),
  '3': ('P3', 'Person'),
}
```

Note that for each tag we also specify the type of the tag, so at the end, assuming that the language has three genders the program will generate the feature (also called parameter in GF):

```
Person = P1 | P2 | P3 ;
```

regardless of the input annotation.

Language Plugins The code generator produces files with specific names following the convention in the GF Resource Grammars Library (Ranta, 2009) (Appendix C). The last three letters of every module name are usually the ISO 3 code of the language which are specified by defining the variable iso3, e.g.:

```
iso3 = "Sqi"
```

If the raw data for the language need cleaning up, the user can define the function preprocess(record), which takes as argument the current record in a source-specific format.

The preprocess function is very low-level while there are common issues which can be handled in a more convenient way. For example, if the function patchPOS(lemma,pos) is defined, it can be used to fix erroneous parts of speech. It takes as input the current lemma and part of speech and returns a new one.

If a lemma must be completely excluded, then the function filter_lemma(lemma, pos) can be defined to return False. Sometimes only specific forms must be excluded which is similarly possible with function filter_form (form,tags).

After the data are cleaned up, the algorithm builds a draft for each paradigm in the form of a tree. The tree can then be manually manipulated by defining one function for every part of speech. For example function patchN(lemma,tree) is invoked for every noun and its corresponding paradigm tree. The tree is a simple Python dictionary from tags to sub-trees. The leafs of the tree are the corresponding word forms. The shape of the tree affects the the final code, which means that these functions are the main way to control how the generated grammar will be structured.

The preference to particular grammar structure is mostly a matter of a design choice, but there is one particular case when this is particularly important. Some of the grammars in the GF libraries are just sketches while other are quite mature. In some languages (Greek, Slovak, Czech, others) we would wish to rebuild the morphology to gain better coverage, while preserving the syntax. In that case the patch functions can be used to shape the trees in the desired way.

Finally, when learning the smart paradigms, the algorithm chooses the principal parts based on statistical measures. On the other hand, every language also has linguistic traditions that we want to respect. For example, the reference form for verbs in Macedonian is not the first person, singular but third person. In situations like this we can hard-code the desired principal parts, while the algorithm fills in the rest. This is done with a configuration like this:

```
required_forms = {
  'N': ['s;Indef;Sg', 's;Indef;Pl'],
  'V': ['present;Sg;P3'],
  'A': ['s;Indef;Masc', 's;Indef;Fem', '
    adverb']
}
```

C. Examples of output of each module

C.1. Output of module 1

```
param Species = Indef | Def ;
param Case = Nom | Acc | Dat | Ablat ;
param Number = Sg | Pl ;
param Gender = Masc | Fem ;

oper Noun = {s: Species => Case => Number => Str; g: Gender} ; -- 3978
oper mkNoun : (_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_ : Str) -> Gender -> Noun =
       \f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,f15,f16,g ->
          { s = table {
                  Indef => table {
                             Nom => table {
                                      Sg => f1 ;
                                      Pl => f2
                                    } ;
                             Acc => table {
                                      Sg => f3 ;
                                      Pl => f4
                                    } ;
                             Dat => table {
                                      Sg => f5 ;
                                      Pl => f6
                                    } ;
                             Ablat => table {
                                        Sg => f7 ;
                                        Pl => f8
                                      }
                           } ;
                  Def => table {
                           Nom => table {
                                    Sg => f9 ;
                                    Pl => f10
                                  } ;
                           Acc => table {
                                    Sg => f11 ;
                                    Pl => f12
                                  } ;
                           Dat => table {
                                    Sg => f13 ;
                                    Pl => f14
                                  } ;
                           Ablat => table {
                                      Sg => f15 ;
                                      Pl => f16
                                    }
                         }
                } ;
            g = g
          } ;
```

Listing 1: A function in ResSqi.gf which builds a noun in Albanian from a list of all forms.

C.2. Output of module 2

```
mkN006 : Str -> N ;
mkN006 base =
  case base of {
    base_1+"ll" => lin N
      { s = table {
```

```
            Indef => table {
                      Nom => table {
                               Sg => base_1+"ll" ;
                               Pl => base_1+"je"
                             } ;
                      Acc => table {
                               Sg => base_1+"ll" ;
                               Pl => base_1+"je"
                             } ;
                      Dat => table {
                               Sg => base_1+"lli" ; --guessed
                               Pl => base_1+"jeve" --guessed
                             } ;
                      Ablat => table {
                                 Sg => base_1+"lli" ; --guessed
                                 Pl => base_1+"jesh" --guessed
                               }
                    } ;
            Def => table {
                     Nom => table {
                               Sg => base_1+"lli" ;
                               Pl => base_1+"jet"
                             } ;
                     Acc => table {
                               Sg => base_1+"llin" ; --guessed
                               Pl => base_1+"jet" --guessed
                             } ;
                     Dat => table {
                               Sg => base_1+"llit" ; --guessed
                               Pl => base_1+"jeve" --guessed
                             } ;
                     Ablat => table {
                               Sg => base_1+"llit" ; --guessed
                               Pl => base_1+"jeve" --guessed
                             }
                   }
          } ;
      g = Masc
    };
  _ => error "Can't apply paradigm mkN006"
} ;
```

Listing 2: A Paradigm function in MorphoSqi.gf with takes a lemma and produces all forms

C.3. Output of module 3

```
regN : Str -> N   -- s;Indef;Nom;Sg
  = \form -> case form of {
  _ + "lth" => mkN001 form;
  _ + "kth" => mkN206 form;
  _ + "eth" => mkN005 form;
  _ + "gth" => mkN005 form;
  _ + "ath" => mkN031 form;
  ...
  _ + "sh" => mkN017 form;
  _ + "xh" => mkN034 form;
  _ + "dh" => mkN013 form;
  ...
  _ + "h" => mkN002 form;
  _ + "t" => mkN002 form;
  _ + "d" => mkN002 form;
  _ + "r" => mkN031 form;
  ...
```

```
18      } ;
```

Listing 3: A smart paradigm function for nouns which guesses and executes an inflectional class