Abstract syntax as interlingua: Scaling up the grammatical framework from controlled languages to robust pipelines

Downloaded from: https://research.chalmers.se, 2026-09-12 23:57 UTC

Citation for the original published paper (version of record):
Ranta, A., Angelov, K., Gruzitis, N. et al (2020). Abstract syntax as interlingua: Scaling up the grammatical framework from controlled languages
to robust pipelines. Computational Linguistics, 46(2): 425-486.
http://dx.doi.org/10.1162/COLI_a_00378

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

Abstract Syntax as Interlingua: Scaling Up the Grammatical Framework from Controlled Languages to Robust Pipelines

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

Krasimir Angelov
Chalmers University of Technology
University of Gothenburg
Department of Computer Science
and Engineering
Krasimir.Angelov@cse.gu.se

Normunds Gruzitis
University of Latvia
Institute of Mathematics and
Computer Science
normunds.gruzitis@lumii.lv

Prasanth Kolachina
Chalmers University of Technology
University of Gothenburg
Department of Computer Science
and Engineering
kolachina.prasanth@gmail.com

Abstract syntax is an interlingual representation used in compilers. Grammatical Framework (GF) applies the abstract syntax idea to natural languages. The development of GF started in 1998, first as a tool for controlled language implementations, where it has gained an established position in both academic and commercial projects. GF provides grammar resources for over 40 languages, enabling accurate generation and translation, as well as grammar engineering

Submission received: 21 March 2019; revised version received: 1 December 2019; accepted for publication: 29 January 2020

https://doi.org/10.1162/COLI_a_00378

© 2020 Association for Computational Linguistics
Published under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) license

tools and components for mobile and Web applications. On the research side, the focus in the last ten years has been on scaling up GF to wide-coverage language processing. The concept of abstract syntax offers a unified view on many other approaches: Universal Dependencies, WordNets, FrameNets, Construction Grammars, and Abstract Meaning Representations. This makes it possible for GF to utilize data from the other approaches and to build robust pipelines. In return, GF can contribute to data-driven approaches by methods to transfer resources from one language to others, to augment data by rule-based generation, to check the consistency of hand-annotated corpora, and to pipe analyses into high-precision semantic back ends. This article gives an overview of the use of abstract syntax as interlingua through both established and emerging NLP applications involving GF.

1. Introduction

Like many other computing tasks, Natural Language Processing (NLP) can be divided to two phases: analysis and synthesis. The analysis phase involves Natural Language Understanding (NLU), whereas the synthesis phase may serve different kinds of tasks, such as Natural Language Generation (NLG), question answering, and conversion to logical forms. Machine Translation (MT) is a representative example of NLP, which can be structured as a combination of NLU from one language with NLG to another language. This architecture is shown in the Vauquois triangle (Figure 1, adapted from Vauquois 1968): The result of NLU is a meaning representation in an **interlingua**, and NLG renders this meaning faithfully in the target language.

For Vauquois, interlingua-based translation was the ideal, because it guarantees that the source language meaning is rendered faithfully in the target language, whatever differences there are in the surface syntax. At the same time, he was aware that this method might not be realistic to implement in practice, and that approximations might be needed. Thus his triangle includes shortcuts on different levels: word-to-word transfer and syntactic transfer. Character-to-character transfer is a later approach used in Neural Machine Translation (NMT; Lee, Cho, and Hofmann 2016).

We have annotated Figure 1 with labels indicating how problematic each phase is. On the lowest level, lexical (morphological) analysis is well understood thanks to a long tradition and reliable analyzers; exceptions are provided by languages that do not mark word boundaries and by systems that have to recover from spelling mistakes. Going one step higher, syntactic parsing, which we take to include part-of-speech disambiguation, is uncertain, because natural languages are ambiguous and also because syntactic grammars are usually incomplete. For similar but even stronger reasons, semantic interpretation can be labeled as highly uncertain. Finally, the semantic interlingua itself is problematic, and all known attempts to build one have remained incomplete.

Target generation is in principle a solved problem if it starts from a formalized interlingua, since it can be performed by a deterministic program. But the transfer from source to target is less certain on the lower levels than on the higher levels; for example, on the word-to-word level, the order of words in the target language is just a guess.

Most traditional MT methods operate on the lower levels: Statistical MT (SMT; Koehn 2010) is usually either word-to-word or phrase-to-phrase transfer,[1] and so are

1 A "phrase" in SMT can be any sequence of words, not necessarily a grammatical phrase.

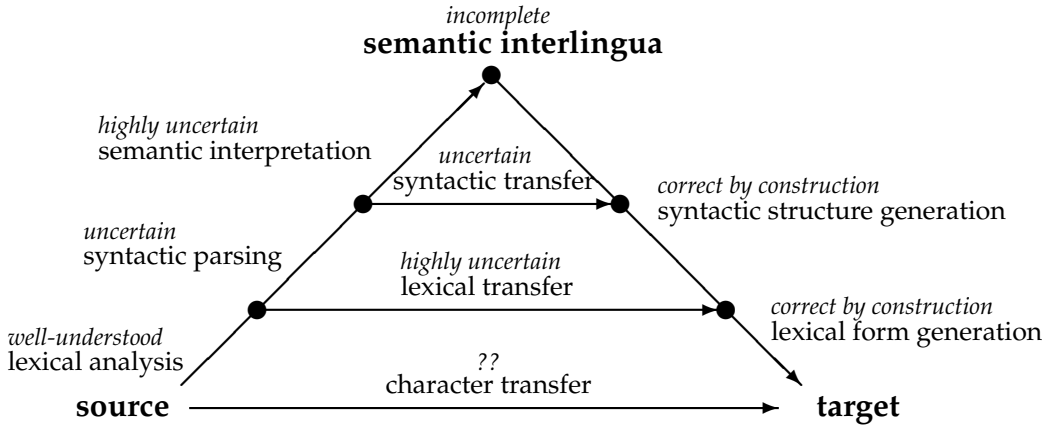


Figure 1
The Vauquois triangle, annotated with estimates of how difficult or how reliable each translation phase is.

rule-based systems such as Apertium (Forcada et al. 2011). More recently, NMT has introduced character-to-character transfer, which Vauquois did not consider at all (Lee, Cho, and Hofmann 2016) . But in a system where all steps are uncertain, ranking the alternatives globally rather than separately in a pipeline actually does make sense, even though it comes with the price of not being able to use linguistic knowledge and to explain step by step how the translation is achieved.

At the same time as going down to the lowest possible level of transfer, NMT has made claims of using an interlingua, just not of the kind that humans construct but something that arises as an internal representation in a neural network (Lu et al. 2018). The main advantage is the same as in traditional interlinguas: one does not need to build $n(n-1)$ translation functions to cover all pairs of n languages, but it is enough to have $2n$ (Figure 2). In the NMT world, this method has been given the name **zero-shot translation**—translation between language pairs for which the system has not been explicitly trained. Similar advantages can be shown in other NLP tasks: For instance, the answer function in question answering can be defined uniformly in terms of the interlingua, rather than for all languages separately (Zimina et al. 2018).

A universal interlingua that accurately expresses all meaning in natural languages would be the ideal translation method. But no one has managed to build one, and it is not sure if the concept even makes sense. Therefore it is useful to consider interlinguas with less ambitious specifications:

- **deep interlinguas**, covering fragments of language with high precision,
- **shallow interlinguas**, with wide coverage but lower precision, and
- **layered interlinguas**, using shallow interlinguas as backup for deep ones.

Examples of the first kind can be found in **Controlled Natural Languages** (CNL), where the interlingua can be a domain-specific semantic structure or an "ontology"; see Kuhn (2014) for a recent survey of CNL, and Hutchins and Somers (1992) for its potential in translation. A deep interlingua can actually start with a semantic model and generate a

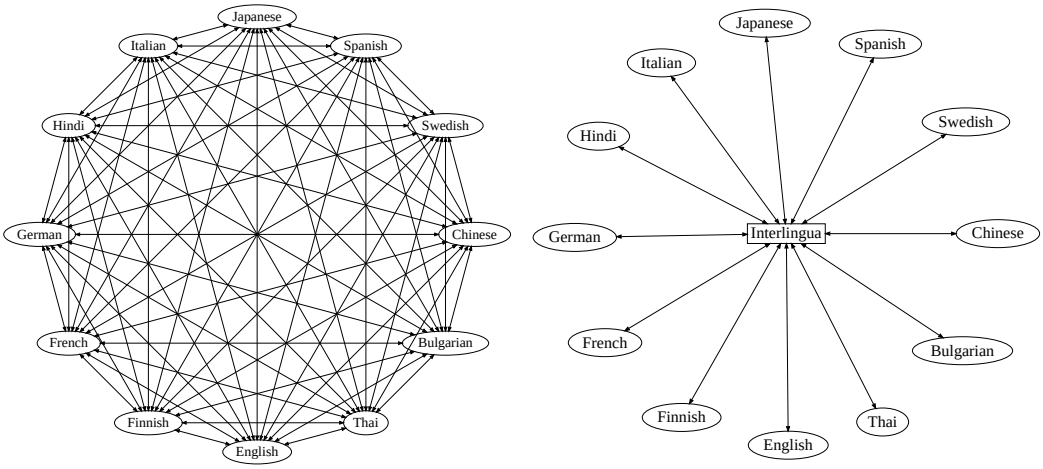


Figure 2
A translation system with 12 languages using transfer (left) vs. interlingua (right).

fragment of language that matches it, rather than starting with a language and trying to find a semantic model.

Shallow interlinguas are representations of some aspects of language that help translation, without going as deep as the full meaning to be rendered. A simple example of a shallow interlingual representation would be a sequence of word senses, such as the ones in multilingual WordNets. The translator would first map source language words to these senses, and then map the senses as target language words in the same order. The words would often appear in wrong senses, in wrong order, or in wrong forms, but the translation could still be helpful.

A bit less shallow interlingua could add syntactic structure to the words, so that it could reorder the words or even enforce agreement in accordance with the rules of the target language. Such a system would still not always select correct word senses, identify multiword constructions, or change the syntactic structure when needed (Dorr 1994). But it could guarantee the grammatical correctness of the output.

Figure 3 shows the architecture of a layered interlingua system, where the transfer-based shortcuts of the original Vauquois triangle are replaced by interlinguas of varying depth. Such a system enjoys some of the advantages of the original interlingua idea—in particular, the linear scale-up as languages are added.

The main topic of this article is to show how to build and combine interlinguas of different levels, by using **Grammatical Framework** (GF; Ranta 2004b, 2011a), a tool designed for this very purpose. More precisely, GF is a special-purpose programming language for defining interlinguas (**abstract syntaxes**) and reversible mappings from them to individual languages (**concrete syntaxes**). GF has no single interlingua or fixed set of languages, but a set of tools for building new ones and reusing old ones.

GF was originally intended as a tool for CNL implementations, and it has been widely used in both academic and commercial CNL projects. But many research efforts in the last ten years have been devoted to scaling up beyond CNL. This has often happened by combining GF grammars with other approaches.

This article is an overview of the interlingua idea as implemented in GF, with focus on recent wide-coverage work.

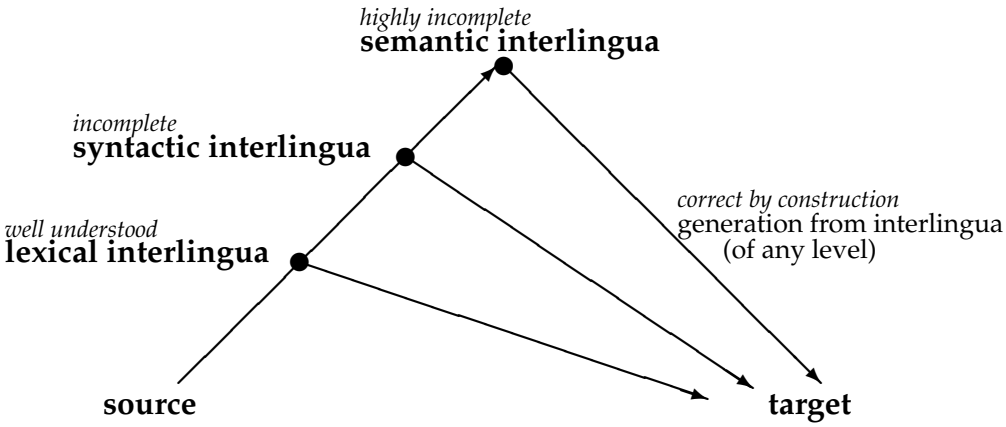


Figure 3
The Vauquois triangle, with transfer shortcuts replaced by interlinguas on various levels.

- Section 2 outlines the background of GF and its development from CNL to open domain tasks.
- Section 3 gives a quick tutorial on GF, with a focus on how linguistic variation can be modeled in terms of abstract and concrete syntax.
- Section 4 gives a summary of GF's **Resource Grammar Library** (RGL), which aims to formalize the main syntactic structures of the world's languages and plays a major role in both CNL and wide-coverage applications.
- Section 5 shows how NLP systems can be built by combining interlinguas of different levels, ranging from chunks of words to logical formulas.
- Section 6 relates GF to dependency parsing, in particular showing the close correspondence between RGL and Universal Dependencies (UD).
- Section 7 summarizes recent work where GF is related to other approaches: WordNet, FrameNet, Construction Grammar, and Abstract Meaning Representation (AMR).
- Section 8 concludes and outlines suggestions for future work.

Table 1 shows how Sections 2–7 depend on each other.

2. The Background and Evolution of GF

Grammatical Framework (GF) was born at Xerox Research Centre Europe in 1998 within the project *Multilingual Document Authoring* (Dymetman, Lux, and Ranta 2000; Mäenpää and Ranta 1999). The target was Controlled Natural Language (CNL): precisely defined language fragments for specific domains. The first domains addressed were pharmaceutics, mathematics, and tourist phrasebooks.

GF was expected to enable high-quality translation via semantic interlinguas. The semantic interlingua would define the meaning to be preserved in translation. The translation itself would be done by parsing the source language into the interlingua,

Table 1
Dependencies between Sections 2 to 7. Sections 2, 3, 4 should be read in sequence, while 5, 6, 7 only depend on these and not on each other.

Section	Topic	Prerequisites
Section 2	GF background	n/a
Section 3	GF tutorial	Section 2
Section 4	RGL	Section 2, 3
Section 5	Levels of interlingua	Section 2, 3, 4
Section 6	UD and GF	Section 2, 3, 4
Section 7	Other approaches	Section 2, 3, 4

and generating the target language from it. Such systems had been built before GF, but they were considered expensive to develop and maintain. Rosetta (1994) was perhaps the most comprehensive system of this kind, using semantic structures inspired by Montague grammar (Montague 1974) as interlingua.

The mission of GF was to make it cheaper to build multilingual CNL systems. This was to be achieved by developing a declarative, high-level formalism, from which the actual processing components (parsing, generation, interactive editor) could be derived automatically. The overall design of GF software was inspired by another system previously developed at Xerox: the Xerox Finite State Tool (XFST; Beesley and Karttunen 2003). The common features include

- a high-level programming language for grammar writers,
- a low-level but efficient run-time format,
- automatic compilation from high to low level,
- mathematical theory underlying compilation and run-time algorithms,
- neutrality with respect to linguistic theories, and
- development tools for grammar engineering.

In XFST, the high-level language is regular expressions, extended with abstraction mechanisms such as user-definable macros and built-in custom operators. The compiler converts this language to finite state transducers, for which there are portable run-time systems that can be embedded in NLP pipelines.[2]

GF needed more expressive power than finite state transducers, but the same design principles were followed to make the formalism portable, adaptable, and supported by a mathematical metatheory. The starting point was to build GF on top of a **Logical Framework** (LF; Harper, Honsell, and Plotkin 1993), based on the Constructive Type Theory of Martin-Löf (1984). The LF origin also explains the name:

$$GF = LF + \textit{grammar rules}$$

LF had a purpose analogous to GF: declarative definition of different logics, such as classical, constructive, temporal, modal, dynamic, and so forth. In computerized

2 Ranta (2019) contains more comparisons between GF and XFST.

mathematics and in software verification, it is often better to work with such special-purpose logics than with one monolithic logic. Implementing them became much more feasible as a generic framework provided algorithms for inference, proof checking, and proof search, together with a notation supporting declarative definitions of logics.

The price to pay for a uniform treatment of logics in LF was that the notation was likewise uniform. The basic framework used prefix notation and ASCII identifiers for all logical operators. For instance, the conjunction of A and B is expressed

```
(Conj A B)
```

similarly to the LISP programming language. Later versions of LF introduced "pretty-printing rules" to convert such formulas to more user-friendly notations—for instance,

```
(Conj X Y) ==> X & Y
```

The grammar rule notation of GF arose as a generalization of such pretty-printing rules, adding mechanisms that enable the definition of natural language structures.

In GF, the LF part was to be used for interlinguas, whereas grammar rules would define relations between interlinguas and natural languages (English, French, Finnish, etc.). The grammar rules would be powerful enough to define formal languages as well, such as logics and query languages; one important application area of GF has indeed been translations between formal and natural languages (Johannisson 2005; Ranta 2011b). The interlingua/LF part in GF was called **abstract syntax**, and the language parts **concrete syntaxes**, following familiar compiler terminology. A GF grammar consists of one abstract syntax and any number of concrete syntaxes, and is thus inherently a **multilingual grammar**.

The abstract syntax can be vastly different in domains ranging from mathematics to tourist phrasebooks. However, the linguistic mechanisms needed in the concrete syntax—morphology, agreement, word order—are largely the same in all areas of discourse. It would be tedious to implement these mechanisms over and over again in different applications. The solution to this problem was to develop a *library* of basic linguistic structures. The library, called the **Resource Grammar Library** (RGL), started to develop in 2001. It was inspired primarily by the Core Language Engine (Alshawi 1992), where a similar need had been identified. It is also related to Xerox ParGram (Butt et al. 2002) and the LiNGO Matrix (Bender and Flickinger 2005), which are collections of grammars for different languages.[3]

The RGL implements morphology and syntax rather than semantics. It was not meant to be a translation interlingua in itself, but to boost the development of domain-specific semantic CNLs by taking care of linguistic details. Not only does it save the work of implementing morphology and syntax again and again. It also has its own common abstract syntax, which provides a common Application Programming Interface (API) enabling a CNL implementation for many languages simultaneously, sharing almost all of the source code except content words (see Section 5).

But does GF scale up to mainstream NLP tasks? Its original mission is expressed in the diagram of the coverage/precision trade-off (Figure 4). Although it seems impossible to achieve complete coverage and complete precision at the same time, it is possible to opt for either of them and make progress by improving the other. Much of

3 Incidentally, GF started at Xerox about the same time as ParGram, but their goals were different, with GF focusing on interactive document authoring, ParGram on more traditional translation.

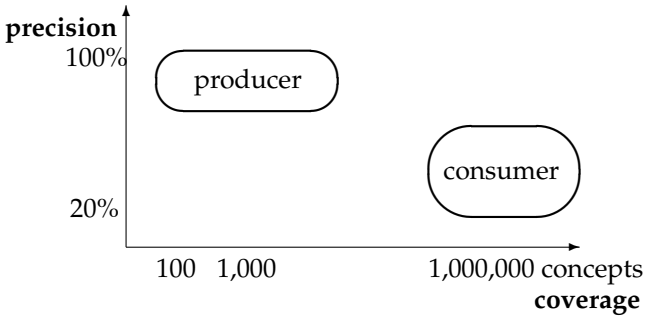


Figure 4
The precision–coverage trade-off, as correlated with producer vs. consumer translation tasks. High precision can be achieved in tasks where a producer of content wants to translate a limited kind of content. High coverage is needed when consumers of content want to translate any kind of content.

NLP opts for 100% coverage, but tries to improve precision, as measured, for instance, by the BLEU score in translation (Papineni et al. 2002). The state of the art in machine translation achieves around 50% or less, depending on language pair, as shown by the latest Workshop on Machine Translation (WMT) evaluations.[4] The philosophy of GF has been orthogonal to this: Maintain full precision, and develop by increasing coverage.

Figure 4 relates the coverage–precision trade-off to typical NLP tasks. High coverage is needed in "consumer" tasks, where the input is unpredictable. The system must be able to cope with millions of "concepts," such as translation units (e.g., words and multiwords). High precision, in contrast, is needed in "producer" tasks, such as the publication of some content in different languages. Human translation is still the state of the art in such tasks, but automation can be possible if the type of expected content can be fixed. The MÉTÉO system (Chandioux 1976) is an early example of this. A typical modern example is e-commerce Web sites selling to multiple countries. The size of such tasks is typically just thousands of concepts, rather than millions.[5]

Producer and consumer tasks are known as **dissemination** and **assimilation**, respectively, in machine translation terminology (Hutchins and Somers 1992). The distinction has gone roughly parallel to the use of symbolic vs. statistical methods. But early MT systems often used symbolic methods for assimilation as well, and this tradition continues in Apertium (Forcada et al. 2011). The advantages of symbolic methods include

- **explainability**: the system can show why it yields a certain result, by for instance showing a syntax tree;
- **programmability**: it is possible to fix bugs without breaking everything else and
- **data austerity**: no large amounts of data are needed.

4 In EuroMatrix evaluation matrix, `http://matrix.statmt.org/`, the best result in Newstest 2019 was 45.2 for English–German.

5 Ranta, Unger, and Vidal Hussey (2015) describe a commercial GF project involving around 1,000 concepts collected from a database about properties of buildings, whereas Section 5.4 discusses a later project involving over 3,000 concepts from the European law.

Data austerity is particularly important for low-resource languages, which may be unlikely ever to have enough data for statistical methods. Writing linguistic rules can then be a way out. Another aspect is the possibility to use additional knowledge: Statistical methods typically learn translation from parallel texts, whereas correct translation can also require knowledge that is impossible to extract from the texts alone (Kay 2017).

It is natural to ask whether GF could be used for wide-coverage tasks by sacrificing some precision but maintaining explainability, programmability, and data austerity. This question has been the focus of much of GF research for the last ten years. It was first enabled by efficient and scalable parsing techniques (Ljunglöf 2004; Angelov 2009; Angelov and Ljunglöf 2014). The first experiments addressed full-scale morphology implementations (Forsberg and Ranta 2004; Détrez and Ranta 2012), hybrid GF-SMT translation (Enache et al. 2012), and multilingual lexicon extraction (Virk et al. 2014; Angelov 2014). Much of the focus has been on machine translation, with an early mobile demo (Angelov, Bringert, and Ranta 2014) and an emphasis on explainability rather than optimized BLEU scores. The prospects for **Explainable Machine Translation** (XMT) are studied in Ranta (2017), and the general name for all the approaches described in this paper could be **Explainable NLP**.

Figure 5 shows the architecture of a GF-based explainable MT system as an instance of **Explainable Artificial Intelligence**: an AI system which, in addition to the output, also gives an explanation of why this output is produced: For instance, that an image is recognized as a cat because it has a tail and whiskers and four legs (Gunning 2017). The user can inspect the explanation to judge whether the output is correct or plausible. Such explanations are clearly desirable in MT, where the user typically knows just one of the languages and cannot judge correctness when only seeing the input and output. The architecture shown in Figure 5 is inspired by the notion of **certifying algorithms** (McConnell et al. 2011), where the explanation, a **certificate**, can be inspected by an independent **checker** program. In the case of GF-based XMT, the certificate is the interlingual representation (abstract syntax tree), which specifies the meaning that has been extracted from the source utterance by the analyzer and rendered in the target language. This tree can be inspected independently by the user, and also translated

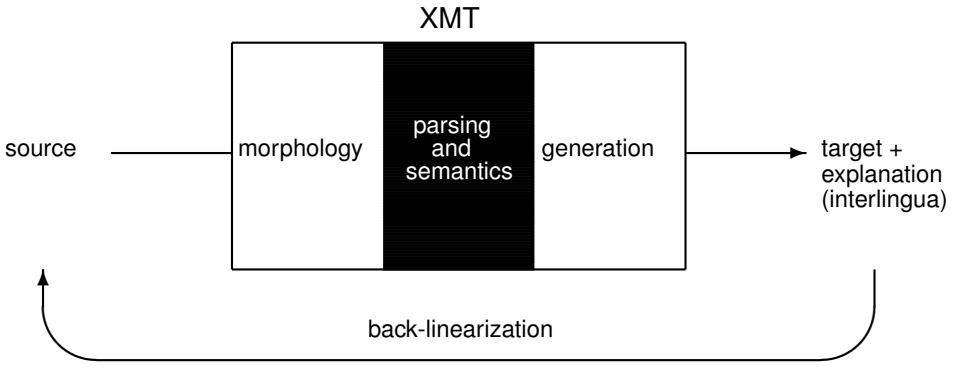


Figure 5
XMT with interlingual representation as explanation. The interlingua can be translated back to the source language for comparison. The "black box" lies between "white" parts that are well performed by grammars. The black part takes care of parser error recovery and semantic disambiguation, where grammars might not be sufficient.

back to the source language by the grammar ("linearization"). Because of this, the tree could actually be produced by a black box such as a neural parser, and still be used with confidence.

3. Modeling Languages with Abstract Syntax

Abstract syntax is a tree representation used in compilers and in programming language semantics. Its purpose is to abstract away from details considered irrelevant for semantics:

- the **shape** of tokens, for example, if one uses `!=` or `<>` as inequality symbol;
- the **order** of constituents, for example, if one uses infix (`2 + 3`), postfix (`2 3 +`) or prefix (`+ 2 3`) notation; and
- the **number** of tokens, for example, if one includes the keyword `then` in "if-then-else" statements or not.

Putting these abstractions together makes it possible to build common abstract syntax representations for languages that look totally different. A typical example is the translation of high-level programming languages to machine languages in a compiler. Figure 6 shows an abstract syntax tree for an arithmetic expression in Java and its translation to Java Virtual Machine assembly language, illustrating variation in the shape, order, and number of tokens.

The place of abstract syntax in a compiler is intermediate between the **front end** and the **back end**. A chief component in the front end is **parsing**, which converts the input source code (string of tokens) to an abstract syntax tree (AST). The back end applies the opposite procedure of **linearization** to convert the AST into a string in the target language. The back end can also involve **semantic analysis** such as mapping the variable symbols into memory addresses or registers, or perform **optimizations** on the code, for instance to minimize the size of the target code or its memory usage. Such back-end operations are implemented by **syntax-directed translation**, which means recursive functions operating on ASTs (Aho et al. 2006).

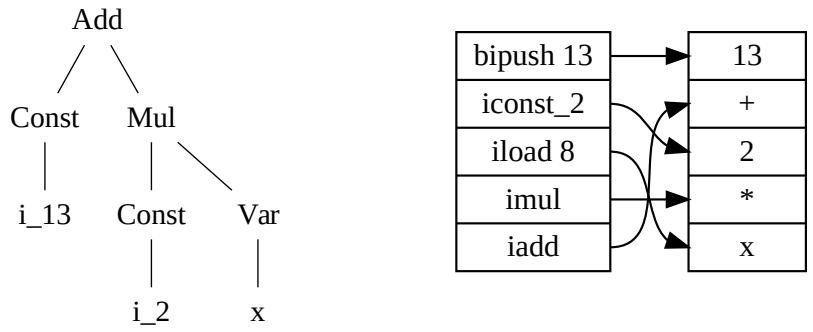


Figure 6
An abstract syntax tree and word alignment for infix to postfix (Java to Java Virtual Machine) translation.

In the early days of compilers, back-end operations were often performed directly in the parser, without explicitly building abstract syntax trees. The purpose was to minimize the usage of the scarce computational resources. **Synchronous grammars** are a method designed for such direct translation, and have also inspired natural language translation (Shieber and Schabes 1990). In modern compilers, building an AST and storing it separately is considered best practice, as it enables more powerful semantic analysis and optimizations (Appel 1998). It also enables sharing compiler components between different source and target languages, because much of the semantic analysis and optimizations can be performed on the AST level. A prime example is the GNU Compiler Collection, which is a "multi-source multi-target compiler" (Stallman 2001).

One reason to use abstract syntax in NLP is similar to compilers: the possibility to share processing components between languages. The back-end part of a compiler corresponds to what in NLP is called **downstream applications**. For instance, linearization corresponds to **surface generation** in systems like question answering, summarization, and interlingual translation. Optimizations share similarities with NLG operations such as aggregation and search of language-specific idioms. Semantic analysis is performed when syntax trees are converted to meaning representations: Doing this on the AST rather than parse trees enables shared syntactic analysis of multiple languages.

The question is now: How well does the AST-centered compiler model apply to NLP? This question has two aspects, which we will consider separately:

- Is it possible to share AST representations between natural languages?
- Can we always interpret informal natural language as formalized ASTs?

The GF was developed primarily to answer the first question: to define systems where shared ASTs are mapped to different natural languages. The answer is amply covered by earlier GF publications, but is summarized here to make this article self-contained and open the way to discussing the second question.

3.1 Modeling Natural Language with Abstract Syntax

Abstract syntax abstracts away from the features listed above: the shape, order, and number of tokens. These features get us started with natural languages as well. Figure 7 shows an example relating English, Dutch, Latin, and Arabic. In this example,

- the words are, obviously, different in each language,
- English and Dutch are SVO (Subject-Verb-Object), Latin is SOV, Arabic is VSO, and
- Dutch uses two words to express the verb "love."[6]

The sentences in Figure 7 also provide examples of phenomena that are not known in compilers but essential for natural language:

- morphological variation,

6 This is not the most common way to express "love" in Dutch, but chosen here for the sake of illustration; particle verbs are otherwise ubiquitous in Dutch, just like in German.

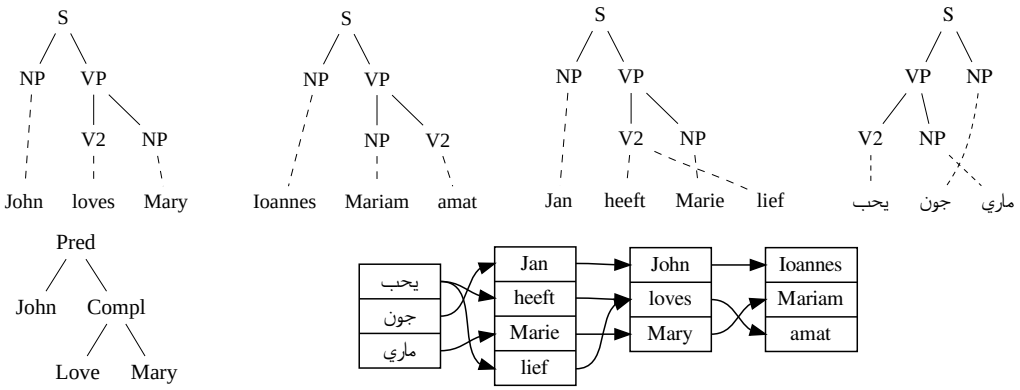


Figure 7
Parse trees for English, Latin, Dutch, and Arabic (above); abstract syntax tree and word alignment (below). Words in the Arabic tree are shown left to right to illustrate their VSO order.

Table 2
Division of a phrase structure grammar for English into abstract and concrete syntax. The division makes it possible to vary the concrete syntax while maintaining the abstract tree structure.

Phrase structure	**Abstract (`fun`)**	**Concrete (`lin`)**
`S ::= NP VP`	`Pred : NP -> VP -> S`	`Pred np vp = np ++ vp`
`VP ::= V2 NP`	`Compl : V2 -> NP -> VP`	`Compl v np = v ++ np`
`V2 ::= "loves"`	`Love : V2`	`Love = "loves"`
`NP ::= "John"`	`John : NP`	`John = "John"`
`NP ::= "Mary"`	`Mary : NP`	`Mary = "Mary"`

- agreement using morphological variation, and
- discontinuous constituents, manifested by crossing branches in parse trees.

The trees and the word alignment in Figure 7 are generated by GF software from the grammar partially outlined in Table 2. We will now explain the main features of GF notation (Sections 3.2, 3.3), and then discuss some issues encountered when modeling the full complexity of natural languages (Section 3.4).

3.2 Abstract Syntax: Categories and Functions

An abstract syntax is defined by a set of **categories** together with a set of **functions** that construct **trees** in these categories. These concepts are implicit in context-free phrase structure grammar, which defines *simultaneously* an abstract syntax and its linearization to a particular language, namely, a **concrete syntax**. The leftmost set of rules in Table 2 uses the BNF (Backus-Naur Form) notation for context-free phrase structure grammars. The middle set of rules is the abstract syntax extracted from this grammar, using the notation of GF, and introducing function names for each BNF rule. The rightmost set of rules is the concrete syntax.

The categories in abstract correspond to the nonterminals of the BNF grammar. The functions correspond to rules, which determine their **types**, with zero or more argument types and one value type. The value type comes from the left-hand sides of the rule, and the argument types from the right-hand side. GF uses the usual notation of functional programming languages, such as Haskell, where the argument types come first and the value type last, all separated by arrows. Thus the `Pred` function

```
fun Pred : NP -> VP -> S
```

can be paraphrased as follows:

The function Pred takes an NP and a VP as its arguments and returns an S as its value.

Applications of functions are written without commas or parentheses, as in `Pred np vp`, but surrounded by parentheses in nested applications.

3.3 Concrete Syntax: Linearization Types, Records, and Tables

Splitting a context-free grammar to abstract and concrete syntax enables us to vary the shape, order, and number of words while keeping the abstract trees constant. In natural language, we also need variations in morphology, agreement, and discontinuous constituents. To enable all this, GF extends concrete syntax with two constructs: tables and records. These constructs are used for defining the **linearization types** of abstract syntax categories: Linearization rules are, technically, functions that operate on the linearization types of abstract syntax categories.[7]

A **table** is a map-like data structure listing alternative items of the same type—for example, inflection tables, as shown in Table 3. Tables operate on **parameter types**, such as number or case, which are defined in concrete syntax separately for each language. Thus Table 3 shows a type of case with two values, representing the nominative and the accusative; a complete Latin grammar would have six cases, a Finnish grammar 15 cases, and a Chinese grammar would not need the case type at all.

A **record** is an object-like data structure (as in object-oriented programming) that stores pieces of information used in combination. One example in Table 3 is the record for the Dutch particle verb *heeft + lief*. Another one is the linearization type of noun phrases (NP), suitable for languages like Latin, which contains an inflection table and a parameter for agreement features.

Tables and records make GF concrete syntax equivalent to **Parallel Multiple Context-Free Grammars** (PMCFG; Seki et al. 1991), as shown in Ljunglöf (2004).[8] This establishes GF as a **mildly context-sensitive** grammar formalism, with polynomial parsing complexity.[9]

7 These functions are moreover **compositional**, in a sense to be explained in Section 3.5.

8 In PMCFG, both tables and records are represented by tuples, but in different ways; see Ljunglöf (2004) for the compilation of GF to PMCFG.

9 A more restrictive notion of mild context-sensitivity is used in the original definition (Joshi 1985), forbidding among other things reduplication.

Table 3
Reading guide for GF notation. The first six rows list the kinds of rules. The rest are expressions for types and objects. The notation $e \Downarrow v$ means that expression e is computed to value v.

Construct	Notation	Example
Abstract syntax category	`cat`	`cat NP`
Abstract syntax function	`fun`	`fun Pred: NP -> VP -> S`
Linearization type	`lincat`	`lincat NP = {s: Case => Str; a: Agr}`
Linearization rule	`lin`	`lin Pred np vp = np.s!Nom ++ vp!np.a`
Parameter type	`param`	`param Case = Nom | Acc`
Auxiliary operation	`oper`	`oper d1 s = table {Nom=>s; Acc=>s+"m"}`
String concatenation	`++`	`"loves" ++ "Mary"`
Token concatenation	`+`	`"Maria" +"m" ⇓ "Mariam"`
Function type	`->`	`NP -> VP -> S`
Function application	`f a b`	`Pred np vp`
Table type	`=>`	`Case => Str`
Table	`table`	`table {Nom =>"Maria" ; Acc =>"Mariam"}`
Selection from table	`!`	`Mary ! Acc ⇓ "Mariam"`
Record type	`{...:...}`	`{s: Str; p: Str}`
Record	`{...=...}`	`{s = "heeft" ; p = "lief"}`
Projection from record	`.`	`Love.p ⇓ "lief"`
Comment	`--`	`-- this is a comment`

3.4 A Linguistic Perspective

We have summarized the main features of GF seen as a programming language. Now we will take a linguistic point of view and see how GF models phenomena like morphology, agreement, and discontinuous constituents. The main point is to show that they can be accurately described in concrete syntax but ignored in abstract syntax.

3.4.1 Inflectional Morphology. The "words" in abstract syntax can be seen as **word senses**, which the concrete syntax can realize by different **word forms**. Thus the abstract function `Love` in Figure 7 corresponds to words in different languages used for expressing a certain sense of the English word *love*. In English, only the form *loves* is needed in this small example, but a full grammar would also cover *love, loved, loving*. In Latin, the corresponding verb *amare* has over 100 forms: *amo, amas, amat, amabam,...*.

To describe a language in isolation from other languages, one could come far with a context-free grammar that has a separate rule for every form of every word. But if we want to share an abstract syntax, we must abstract away from morphological variation. We want to have just one abstract syntax function for each word sense, and leave it to the concrete syntax to list the forms needed in each language. In GF, we do this by generalizing the linearization types (`lincat`) from strings (`Str`) to the table data structure. The tables, as well as the parameters used in them, can be different in every language, so that the abstract syntax can ignore morphological variation.

Actually populating the morphological tables is an engineering effort of its own. The GF programming language provides tools for automating much of this task. A powerful method is **smart paradigms**, which are concrete syntax functions that construct an inflection table from just one or a few forms. For instance, the regular verbs of English can be built with a smart paradigm that inspects the infinitive form and adds slightly different endings depending on the verb, for example, *walk-walks-walked-walking*

as contrasted to *wash-washes-washed-washing* or *cry-cries-cried-crying*. **Regular expression pattern matching** is used for identifying inflection types. For instance, verbs ending with a sibilant produce forms where the third person present ending is *es* instead of *s*:

```
s@(_+("s"|"z"|"sh"|"ch")) => forms s(s+"es")(s+"ed")(s+"ing")
```

A smart paradigm can take more than one form as its arguments to infer the rest. For instance, the five forms of an irregular English verb can usually be built from three, which are the same as in usual descriptions of irregular verbs in dictionaries.

Morphology with smart paradigms was the first example of GF scaling up to full-scale language processing tasks. Détrez and Ranta (2012) performed an evaluation for a few languages, showing that complete inflection tables can usually be inferred from just a bit more than one form on the average, even in languages like Finnish that have hundreds of word forms in total. As a practical benefit, this makes it efficient to build morphological dictionaries from word lists that do not include morphological information. The multilingual WordNet lexicon (Section 7.1) is an example of this.

3.4.2 Derivational Morphology. Inflectional morphology in GF uses tables and smart paradigms. The same mechanisms also work for **derivational morphology**—the formation of new words from given ones. For example, turning an English adjective into a noun with the suffix *ness* can be performed by a smart paradigm that among other things turns *happy* to *happiness*. In Finnish, the corresponding function must make some more distinctions to use the suffix *[u]us*: *paha-pahuus* "bad," *keltainen-keltaisuus* "yellow," *hyvä-hyvyys* "good," *rikas-rikkaus* "rich," and so forth.

It is not enough just to define the morphology of derivational suffixes, because semantically similar derivations may be performed by different suffixes. For example, the noun resulting from *available* is *availability* rather than *availableness*. This information must be recorded in the lexicon, for instance, in a record field attached to each adjective, if a productive rule for adjective to noun conversion is to be included in the grammar.

Unlike inflectional morphology, derivational morphology is not completely productive. The approach usually followed in GF is therefore not to use general abstract syntax functions for things like adjective-noun derivations, but just to provide paradigms that can be used whenever a derivation is mandated by the abstract syntax.

3.4.3 Segmentation and Compounding. Large inflection tables can become unpractical when the lexicon grows. Thanks to smart paradigms, the GF source code can still be kept small. But when their applications are compiled to full tables, run-time grammars can grow very large.

This problem can be solved by treating agglutinated morphemes as separate tokens. For example, Finnish verbs have over 10,000 forms, if all combinations with clitics are taken into account. But they can all be formed by concatenating suffixes to one of 14 stems.[10] Therefore, it is sufficient to store these 14 stems in the inflection tables and add suffixes at runtime. For example, the equivalent of "did we walk" is *kävelimmekö*, which is produced from *käveli+mme+kö* ("walk(past)+1personPlural+question").

10 These stems were identified by analyzing the full inflection tables of a comprehensive set of verbs. For a large majority of verbs, much fewer stems would be enough.

In running Finnish text, the morpheme boundaries are not visible. The parser hence has to restore the boundaries in order to match the original grammar rules. An algorithm for this is presented in Angelov (2015). It is an essential ingredient in enabling GF to parse morphologically rich languages with large lexica.

Segment analysis is also used when forming compound words in languages like Finnish, German, and Swedish. Thus English *summer time* translates compositionally to Swedish *sommar+tid*, which is rendered as *sommartid*. The algorithm of Angelov (2015) separates these tokens when analyzing the Swedish word. In addition, the Swedish lexicon (as well as Finnish and German) must keep track of the special forms that are used in compounding, so that for instance *response time* becomes *svars+tid* and not *svar+tid*.

Segmentation can be ambiguous and resolvable only by clues from syntax and semantics. This is yet another way in which natural languages differ from programming languages: In compilers, segmentation is performed by (usually) deterministic preprocessing known as **lexing**. The algorithm in Angelov (2015) performs segmentation as a part of parsing and can therefore use syntactic clues to resolve ambiguities. It can also use semantic clues, if the abstract syntax encodes them (Section 5). In this sense, GF supports holistic end-to-end processing similar to neural networks, and not only compiler-like pipelines neatly divided into distinct components.

Another example of holistic segmentation is the GF resource grammar for Chinese (Ranta, Tian, and Qiao 2015), where each character is treated as a separate token. Inserting word boundaries is left to the parser, which can therefore use all the clues available in the grammar.

3.4.4 Agreement and Government. In addition to inflection tables, lexical information in a concrete syntax defines **inherent features**, such as the gender of nouns in many languages. An example in Table 3 is the linearization type of noun phrases,

```
lincat NP = s : Case => Str ; a : Agr
```

Agreement is an interplay between inherent and inflectional features. Assuming that verb phrases are tables on the agreement features,[11]

```
lincat VP = Agr => Str
```

we can express the NP–VP agreement in predication, where the inherent features of the NP are passed to the VP:

```
lin Pred np vp = np.s ! Nom ++ vp ! np.a
```

Government is, similarly to agreement, treated in terms of inherent and inflectional features. One example is the case government of two-place verbs, V2. V2 has an inherent case, which determines the case inflection of the complement NP. Besides the case itself, the "complement case" may indicate a preposition, as a part of the lexical information (**valency**) of a verb. The abstract syntax does not specify complement cases, but only the number of complements and their abstract syntax types. This is why we use the

11 VP can also depend on other features such as tense, but we omit the details here.

category symbol `V2` rather than transitive verb (`TV`): transitivity, which governs a direct object case, is a language-dependent feature that can be left to the concrete syntax.

3.4.5 Discontinuous Constituents. GF implements discontinuous constituents as records that contain more strings or inflection tables than just one. Figure 7 shows two examples:

- the Dutch verb for "love," where *heeft* and *lief* can be separated by the object; and
- the Arabic `VP`, which has separate fields for the verb and its complement to enable the VSO order.

In general, a concrete syntax record may have to contain several fields in order to match a given abstract syntax. For example, `VP` in German and Scandinavian can have fields for all of the positions in the **topological structure** (Diderichsen 1962), which permits the reordering of constituents in different syntactic contexts (main clause, question, subordinate clause, topicalizations).

3.4.6 Linguistic Generalizations. Plain PMCFG is a low-level language, where grammar rules are encoded in ways that are unintuitive and repetitive for humans. At the same time, it is well-suited for data-driven parsing, and the models can be directly used in GF's run time. An example is Kallmeyer and Maier (2013), whose model was used in Angelov and Ljunglöf (2014), showing that the run-time performance of a GF parser is competitive even in large-scale statistical parsing.[12]

The source language of GF has a very different look and feel: It follows the design of modern functional programming language with abstraction mechanisms that enable the grammarian to express many kinds of **linguistic generalizations**. The goal has been to give the linguist a "freedom of speech" and help avoid repetitive coding both inside and between languages. Here is a summary of some of these mechanisms:

- The abstract syntax itself, expressing constituent structure in a crosslingual way.
- Parameterized modules (functors), enabling sharing of concrete syntax code within language families. For example, most syntax rules in Romance languages can be uniformly described in shared modules, from which the rules for individual languages are obtained by fixing the values of some parameters (Section 4).
- Regular expression pattern matching, enabling smart paradigms and one-line definitions of lexical items, which list only one or a couple of characteristic forms.
- Function libraries, enabling the composition of grammars on different levels. For instance, the details of NP–VP predication can be expressed in a single function, which is reused in linearizing different logical propositions (Section 5).

12 The original model is in the closely related formalism of Linear Context-Free Rewriting Systems (Vijay-Shanker, Weir, and Joshi 1987), which is easily convertible to PMCFG.

3.5 Compositionality and Semantics

One inspiration for GF was Montague grammar (Montague 1974), which sees syntax as just a preparation for semantics. The semantics is designed to be **compositional**: Every syntactic rule must have a corresponding semantic rule. Compositionality can be defined in a simple and general way: a tree operation is compositional, if its application on every tree is a function of its applications on the immediate subtrees. More formally: a tree operation * is compositional, if, for every abstract syntax function F with n arguments, there is an n-place function F^* such that

$$(Ft_1 \dots t_n)^* \;=\; F^* t_1^* \dots t_n^*$$

This means that every subtree is a meaningful unit, that is, has an independent meaning that is the same in all larger trees of which it is a part.

In addition to semantic rules, Montague grammar has operations that convert syntax trees ("analysis trees") to strings, but these rules are informal and follow no special format. In GF, these rules are formalized as *linearization* functions, which are compositional in the same sense as Montague's semantics. Thus, in the rule

$$\texttt{lin}\ Ft_1 \dots t_n \;=\; \dots t_1 \dots t_n \dots$$

the variables $t_1 \dots t_n$ stand for the linearizations of the subtrees, not for the subtrees themselves. The operations on the right-hand sides of these rules are restricted to a few operations on strings, tables, and records. The important thing is that every subtree is a linguistically meaningful unit, that is, has an independent "linguistic meaning" (built from strings, records, and tables). This implies that the claim of a common abstract syntax is not vacuous: to share an abstract syntax function, a language must have a compositional linearization function for it.

The abstract syntax trees of GF also support Montague-style semantics (Ranta 2004a, 2011b), extended by Bernardy and Chatzikyriakidis (2017) to cover the entire abstract syntax of the GF RGL. It has thereby become usable in pipelines that convert open-domain text to logical formulas (of course partially, since RGL parsing is not total). Because of the common abstract syntax, this semantics is in principle defined for all RGL languages.

GF abstract syntax inherits from LF the full power of **dependent types** and **variable bindings**, which enables a compositional model of anaphora, even between sentences in a text (Ranta 1994). This possibility was in fact the main initial reason to develop a type-theoretical version of Montague grammar in Ranta (1994). It is possible to build abstract syntax trees that cover entire texts, and where variables bound in some sentences can be referenced in other sentences. What is needed is a **higher-order function** that takes two arguments: an initial sentence and a continuation, which is a function forming the rest of the text in the context created by the first sentence:

```
fun MkText : (A : Sentence) -> (Context A -> Text) -> Text
```

Longer texts can be built by iterated applications of `MkText`, where each sentence adds to the context in which the next one is formed. This mechanism has been used, for instance, in a CNL for software specifications (Johannisson 2005), but not yet in large-scale NLP.

The main problem with text grammars is that treating complete texts as parsing units is computationally heavy. In practical implementations, it is often useful to relax

compositionality—for instance, to perform anaphora resolution in a separate pass after grammar-based parsing. The technique of **almost compositional functions** (Bringert and Ranta 2008) enables such programs to be written with minimal effort.

Several applications of GF include a back end in logical semantics or translation to query languages. Examples include multilingual question answering (Zimina et al. 2018) as well as commercial projects on financial contracts and software specifications.[13]

3.6 Limitations of GF

The expressivity of a grammar has two senses:

- **Weak generative capacity**: to define what sentences belong to a language.
- **Strong generative capacity**: to give all sentences an intended structural analysis.

In GF, the focus is clearly on the strong capacity: How to make languages match a given abstract syntax. The use of tables and records has usually proven to be sufficient for this when matching the abstract syntax of GF RGL (Section 4).

A more common problem has to do with the computational resources required by PMCFG-based parsing. If linearization were the only operation needed, it could be performed easily by evaluating expressions by usual functional programming techniques. In particular, tables would rarely need to be expanded to their full forms, because each use of them would need only one form, which could be found by lazy evaluation without expanding the entire table. But the compilation to PMCFG needs to expand the tables at compile time and store them for the run time. Huge tables can sometimes, but not always, be avoided by using segmentation (Section 3.4.3). Complex PMCFG grammars can be problematic both for memory usage and for parsing speed.

The heart of the problem is the static type system of concrete syntax—in particular, that all trees of the same category must have the same linearization type. This means that the linearization type must always cover the worst case, which may be relevant for just one word in the category. To give an example, the Spanish word *contigo* ('with you', familiar 'you') is a fusion of the preposition *con* ('with') and the pronoun *tú* ('you'). The word *conmigo* ('with me') is a similar fusion. No other pronoun, let alone noun phrase, fuses with a preposition. Now, if we want to implement the resource grammar function forming adverbials as prepositional phrases,

```
fun PrepNP : Prep -> NP -> Adv
```

we need to cover the combination of the preposition *con* with these two pronouns. With other prepositions, the pronoun would just follow the preposition in the oblique case, as in *sin ti* ('without you'). But because of *contigo* and *conmigo*, Prep and NP must have a parameter in their Spanish linearization type indicating that these combinations have an exceptional behavior. Because of static typing, this leads to redundancies in a majority of tables.

A related problem due to static typing is **defective paradigms**: words that lack some forms required by the linearization type. A technical solution is to use the GF construct

13 See `http://digitalgrammars.com` for some commercial projects.

`nonExist`, which is returned as a value for nonexistent forms. It is actually a special case of the GF construct `variants`, which can handle words that have several forms matching a given form description. However, as shown in Ljunglöf (2004), `variants` is not a standard PMCFG construct, and its inclusion in GF is not straightforward.

An important limit of GF comes from the (widely held) assumption that mild context sensitivity is sufficient for natural languages (Kallmeyer 2010). This assumption has recently been challenged by an analysis of **hyperbatic constructions** in Latin, which make it possible to interleave any words contained in discontinuous constituents (Spevak 2010). The analysis suggests that such constructions require more expressive power than PMCFG or in fact any mildly context-sensitive formalism.[14]

4. The GF Resource Grammar Library

Is it really possible to map all languages to a common interlingual structure? Rather than arguing about this on *a priori* grounds, GF has approached the question in an experimental way: try to build a common abstract syntax and see how far you get. The result is the GF RGL (Ranta 2009b). Its goal is to cover the morphology and syntax of languages, without going too deep into semantics but trying to formalize what is implicit in the tradition of grammars: that languages tend to have categories such as nouns and verbs, and combination rules such as predication and complementation. The syntactic combinations have a shared abstract syntax in the RGL, whereas morphology is defined separately for each language.

The original goal of the RGL was to serve as a *library* for controlled language grammars, in the sense of a **software library**, as further explained in Section 5 (see also Ranta 2007, 2009a). But it turned out later that the RGL can also be used on its own for wide-coverage translation and parsing. This was done first in a pure GF system (Angelov, Bringert, and Ranta 2014). Later, as the shared dependency structure that was built in UD turned out to be close to the abstract syntax of the RGL, it became possible to combine UD and GF into a robust pipeline (Section 6).

4.1 The RGL Languages

The development of the RGL started in 2001, with grammars for English, Swedish, French, and Russian (Khegai 2006). Finnish was soon added, showing the adequacy of the abstract syntax outside the Indo-European family. By the end of the European MOLTO project (2010–2013),[15] a majority of the 27 official languages of the European Union had been covered. As of 2019, the RGL has "complete" implementations of 35 languages—complete in the sense of covering the common abstract syntax and a set of inflectional morphology paradigms able to produce all word forms. The non-Indo-European languages are Arabic (Dada and Ranta 2006), Basque, Chinese (Ranta, Haiyan, and Tian 2015), Estonian (Listenmaa and Kaljurand 2014), Finnish, Japanese (Zimina 2012), Maltese (Camilleri 2013), Mongolian (Erdenebadrakh 2015), and Thai. At least 20 more languages are under construction, many of them Bantu languages (e.g., Ng'ang'a 2012, Pretorius, Marais, and Berg 2017, Kituku, Nganga, and Muchemi 2019).[16]

14 The argument is presented in a paper submitted by François Hublet (2019), which also suggests an extension of GF by constructs that would enable such rules.

15 `http://www.molto-project.eu`.

16 See `http://www.grammaticalframework.org/lib/doc/synopsis.html` and `https://github.com/GrammaticalFramework/gf-rgl` for an updated view.

More than 60 people have contributed to building the RGL, ranging from masters students to senior professors. The effort of building the RGL for a new language is typically from 2 to 6 person-months, requiring between 1,000 and 3,000 lines of GF source code. Table 4 gives an overview and some statistics of the RGL languages.

Some groups of languages are implemented by using **functors**, also known as **parameterized modules**, which means source code depending on a set of parameters,

Table 4
The complete languages of the GF Resource Grammar as of February 2019. ISO = ISO-639-3 code (so-called B code when this makes a difference). Dict = number of lemmas in the dictionary (500 means that there is just a basic lexicon of around 500 lemmas). Transl = used in wide-coverage translation. Appl = used in applications, ++ means also in commercial applications. Publ = publications available, ++ means international publications in addition to academic theses. LoC = lines of GF code (non-empty, non-comment), excluding the dictionary. The sum of two figures means shared functor+language-specific code. The asterisk * means that there is an additional set of low-level morphological paradigms extracted from external sources. Year = when the main part of the grammar was released.

ISO	Language	Dict	Transl	Appl	Publ	LoC	Year
Afr	Afrikaans	500	—	—	—	1200	2009
Ara	Arabic	500	—	++	++	3300	2006
Bul	Bulgarian	53k	+	++	++	4500*	2008
Cat	Catalan	18k	+	+	—	900+1300*	2006
Chi	Chinese	37k	+	++	++	900	2012
Dan	Danish	500	—	++	—	800+1100	2002
Dut	Dutch	23k	+	++	—	1600	2009
Eng	English	65k	+	++	—	1600	2001
Est	Estonian	84k	+	+	++	2600	2013
Eus	Basque	10k	—	—	—	1400	2018
Fin	Finnish	42k	+	++	++	2900	2003
Fre	French	93k	+	++	—	1100+1300	2002
Ger	German	44k	+	++	+	1900	2002
Gre	Greek	500	—	—	++	2600	2012
Hin	Hindi	36k	+	+	++	800+1100	2012
Ice	Icelandic	500	—	—	+	2800	2015
Ita	Italian	30k	+	++	—	800+1300*	2003
Jpn	Japanese	16k	+	+	++	3100	2012
Lav	Latvian	63k	—	+	++	1300	2011
Mlt	Maltese	4k	—	—	++	4600	2014
Mon	Mongolian	24k	—	—	++	1700	2015
Nep	Nepali	500	—	—	+	2100	2013
Nno	Norwegian(n)	500	—	—	—	600+1100	2016
Nor	Norwegian(b)	500	—	++	—	600+1100	2002
Pes	Persian	500	—	++	++	1200	2013
Pnb	Punjabi	500	—	—	++	1600	2012
Pol	Polish	500	—	++	++	5700*	2010
Por	Portuguese	129k	—	++	+	1000+1300*	2018
Ron	Romanian	500	—	++	++	3800*	2009
Rus	Russian	136k	+	—	++	3000	2001
Snd	Sindhi	500	—	—	+	1500	2013
Spa	Spanish	43k	+	++	+	800+1300*	2003
Swe	Swedish	111k	+	++	++	700+1100	2002
Tha	Thai	44k	+	+	—	600	2011
Urd	Urdu	15k	—	+	++	800+1100	2012

from which the compiler produces different run-time code via different values given to the parameters. The most comprehensive functor covers five Romance languages—Catalan, French, Italian, Portuguese, and Spanish. Functors are also used for Scandinavian (Danish, Norwegian Bokmål and Nynorsk, Swedish) and for Hindustani (Hindi, Urdu). In all these cases, around 90% of the code for syntax is shared within the family. A functor is also used in the emerging Bantu grammars.

No effort is known to us trying to share the morphology or the lexicon, even though this could be an interesting typological exercise. It could also be an interesting typological task to create hierarchies of functors, such as Germanic on top of Scandinavian. However, most RGL developers have had ultimately practical goals and used functors only when they clearly ease the effort of language implementation or maintenance. The practical benefit of functors is that they enable adding rules and fixing bugs simultaneously in many languages.

4.2 The Economics of Grammar Writing

Grammar writing can be contrasted with another way to create language resources: by annotating corpora to create data for machine learning (Sharoff and Nivre 2011). A common view is that corpus annotation is "cheaper," in terms of both time and skills required. In our experience, writing a resource grammar indeed requires more programming skills than corpus annotation, and the human labor is therefore "more expensive." But grammarians usually see it as an enjoyable intellectual challenge, which they do voluntarily. It has attracted people who would probably not volunteer for large-scale corpus annotation—for instance, professors of mathematics and computer science.

Grammar writing vs. corpus annotation is not an exclusive choice. Using a grammar-based parser as the first step for building a treebank is a proven method (Marcus, Santorini, and Marcinkiewicz 1993; Oepen and Manning 2004). Because natural language grammars typically yield ambiguous results and have limited coverage, a human is needed to disambiguate and complete the parser output. But this method can still save work compared with fully manual annotation. It can also improve the consistency of the resulting treebanks, by mechanically analyzing the same structures always in the same way.

What is more, a computational grammar is in many ways a richer resource than an annotated corpus. A GF grammar, in particular, gives

- an accurate method to generate language and not only to analyze it,
- a precise definition of crosslingual relations via the shared abstract syntax.

We will see in Section 6.3 that a resource grammar can even *replace* much of the work of hand-annotating corpora, because it enables the generation of synthetic corpora that come a long way in providing data for machine learning without any additional cost.

So what does it cost to build a GF grammar? For a complete resource grammar (in the sense of Table 4), a typical development time is 2 to 6 months of one person's work (**?**). This is an appropriate size for a master's thesis; examples include Japanese (Zimina 2012), Greek (Papadopoulou 2013), Maltese (Camilleri 2013), and Icelandic (Traustason 2016). To extend this to a shallow wide-coverage grammar, such as used in Angelov, Bringert, and Ranta (2014), just a few weeks more is needed, if the lexicon can be extracted from available resources (cf. Sections 5.2 and 7.1). However, there is no GF experience yet from a wide-coverage deep and precise grammar comparable

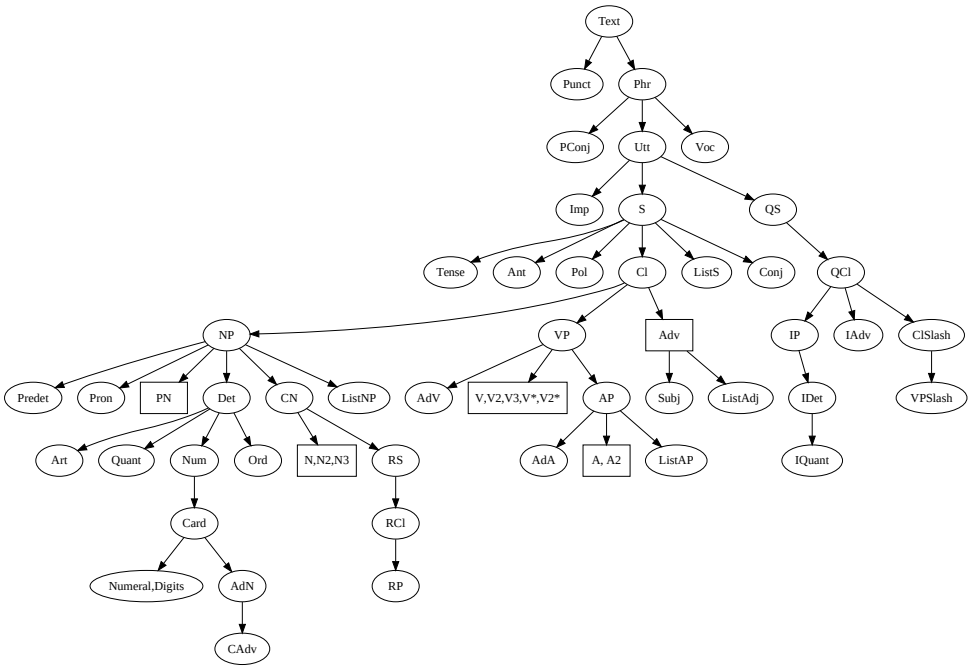


Figure 8
The categories of the common abstract syntax of the RGL, showing their main dependencies. The full dependency graph has several cycles. The rectangles show lexical categories subcategorized by their complement lists, such as V2 for two-place verbs taking one NP complement. V* and V2* mean sets of verb categories taking other complements as well, such as VP, AP, or S.

to the head-driven phrase structure grammar (HPSG) English Resource Grammar (Flickinger 2010).

4.3 The Common Abstract Syntax

The common abstract syntax of the GF RGL has a set of categories, summarized in Figure 8. Figure 9 shows an example that uses many of the categories. Some of the functions combining these categories are shown in Table 5. The table shows all of the RGL functions that are used in the examples of this paper and serves therefore as a quick reference. The total figures are (rounded to nearest tens):

- 90 categories,
- 200 combination functions,
- 140 function words (pronouns, determiners, numerals, etc.),
- 350 content words in a test lexicon.

The common abstract syntax also features a content word test lexicon, designed for testing the morphology and syntax. It includes the Swadesh list (Swadesh 1955) as well as a set of common modern words.[17] As shown in Table 4, many languages also have a

17 The Swadesh list has around 200 lemmas. The modern words are selected from elementary language teaching material with no particular source.

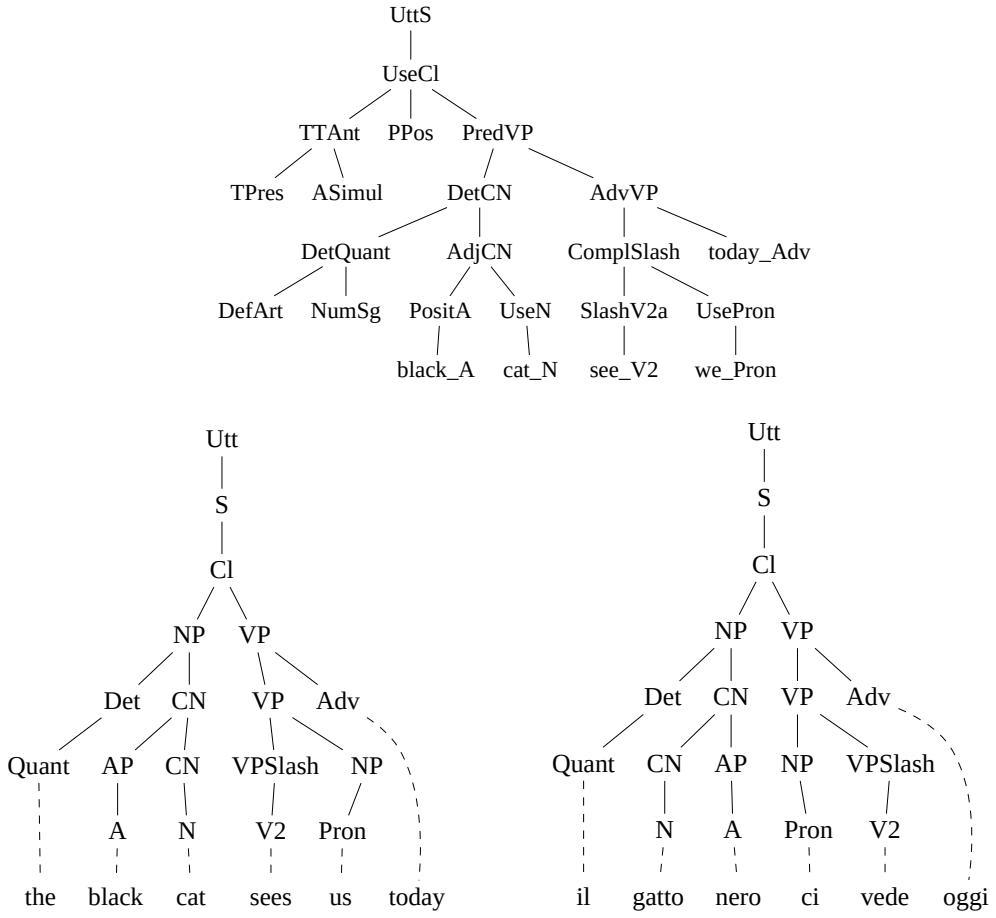


Figure 9
An example of RGL abstract syntax trees (above), with corresponding phrase structure trees for English and Italian (below). The phrase structure trees show two differences in word order: AP–CN and V2–Pron.

larger lexicon, which is either just a monolingual morphological lexicon, or also gives links to other languages by implementing a common abstract syntax of word senses; languages of the latter kind are those that have been used in wide-coverage translation.

Figure 8 and Table 5 suggest that the RGL follows a Western grammar tradition, as formalized by Chomsky (1957) and later in different grammar formalisms in numerous variations. The most important inspiration at the early stage was Montague's "PTQ grammar" ("Proper Treatment of Quantification in Ordinary English," Montague 1974) and its extension in the Core Language Engine, particularly as described in Rayner et al. (2000). The mechanism of "slash categories" to deal with "wh movement" was inspired by Generalized Phrase Structure Grammar (Gazdar et al. 1985), although in a restricted form conformant to PMCFG.[18]

18 The restriction is that each category must correspond to a tuple of a fixed size. Similar reasons made it impossible to reproduce the full power of Montague's "quantifying in" rules, in which an unlimited

Table 5
A sample of the syntactic combination functions in the common abstract syntax, covering the functions used in the examples in this paper. The "UD labels" column refers to the dependency label associated with each argument place of the function, to be discussed in Section 6.

Function	Type	Example	UD labels
AdAP	AdA → AP → AP	*very warm*	advmod head
AdjCN	AP → CN → CN	*big house*	amod head
AdvVP	VP → Adv → VP	*sleep here*	head advmod
ComplSlash	VPSlash → NP → VP	*love it*	head obj
ComplVV	VV → VP → VP	*want to run*	head xcomp
DetCN	Det → CN → NP	*these men*	det head
DetQuant	Quant → Num → Det	*these five*	head nummod
PositA	A → AP	*black*	head
PredVP	NP → VP → Cl	*John walks*	nsubj head
SlashV2a	V2 → VPSlash	*love (it)*	head
UseCl	Temp → Pol → Cl → S	*she won't have slept*	aux advmod head
UseN	N → CN	*cat*	head
UsePron	Pron → NP	*we*	head
UttS	S → Utt	*she sleeps*	head

The main proof for the abstract syntax is that "it works"—at least, has been found to work so far. No claims are made that it should continue to work for all languages of the world and be "universal" in the most pregnant sense of the word. However, the recent advances of UD (Nivre et al. 2016) have not only shown that it is possible to assign the same syntactic structures to vastly different languages, but also ended up with structures very similar to the RGL (Kolachina and Ranta 2016). We will return to the GF-UD correspondence in Section 6.

4.4 Dealing with Structural Differences

The abstract syntax of the GF Resource Grammar is, in the technical sense, an interlingua, which implements a certain linguistic theory of crosslingual syntax. However, it is not good enough to serve as a semantic interlingua that guarantees meaning-preserving translation. The reason is that translation often has to change the syntactic structure to render the original meaning. Here are some simple and well-known examples:[19]

- English *I am a teacher*, French *je suis professeur*: no indefinite article in French.
- English *I like this*, Italian *questo mi piace* ("this pleases me"): swap of subject and object.

number of slots must be available for pronouns corresponding to a bound variable. The full power of Chomsky's transformations is beyond the reach of PMCFG as well. Slash propagation in GF is implemented as function composition, following the tradition in categorial grammars; cf. Steedman (1988).

19 Many more can be found in, e.g., Dorr (1994) and Kay (2017).

- English *there is money*, German *es gibt Geld* ("it gives money"), Swedish *det finns pengar* ("money is found"), Finnish *rahaa on* ("money is"): existentials expressed in numerous ways.
- English *he swims across the river*, French *il traverse la rivière en nageant* ("he crosses the river by swimming"): the mode and direction of movement distributed in different ways.
- English *my name is John*, German *ich heiße John* ("I have-name John"), French *je m'appelle John* ("I call myself John"): different constructions where even the subject changes.

The RGL could of course in principle cope with all this by adding more and more constructions on a more abstract level. From the above examples, the existentials are actually handled in the RGL by special functions that enable compositional translation. But in general, there are too many constructions to keep track of, and special domains of language use come with their own constructions. For example, English *be subject to X* can in legal language be expressed in French as *faire l'objet de X* ("make the object of X"; see Table 7 later in this article for some more examples). The strategy adapted for this is to use domain-specific **application grammars** for precision translation tasks. The RGL is then used as a **library** for building these grammars, so that the authors of application grammars (such as lawyers) do not need to write linearization rules on the level of records and tables, but can use RGL functions to take care of linguistic details (Ranta 2009a).

Using RGL as a library, rather than as a run-time grammar, was its original purpose. The users of the library would only need to know its abstract syntax, and leave the details of linearization to the linguists who have implemented the RGL. We shall return to the library use of the RGL in Section 5, where we look closer into **grammar composition**: a way to combine grammars of different levels (semantics, syntax) by function composition.

4.5 Extensions of the RGL

The common abstract syntax of the RGL can be seen as an *intersection* of the structures available in different languages. There is no claim that it covers all the structures of all languages: On the contrary, the library has a standard set of modules for language-specific extensions. These were originally defined in separate modules for each language or family (where, for instance, Romances language have a common set of extensions).

Later development has led to a "matrix" of extensions, which is the *union* of all language-specific functions and their optional implementations in other languages. Each language has a column indicating the concrete syntax of each such function, either by a genuine rule or by a paraphrase. A typical example is the `ProDrop` function, which drops the string part of a subject pronoun but retains its agreement features. It has genuine entries for some languages (Italian, Spanish, Finnish), whereas languages that do not have pro-drop are just given a paraphrase—an identity function to the effect that the pronoun is not dropped.[20]

20 In Finnish, pro-drop is restricted to the first and second persons, which is of course a part of the linearization rule.

4.6 Limitations of the RGL

The RGL was originally designed to cater to the needs of NLG. For this purpose, it is enough to express all desired content in one way: One does not need to cover all possible ways to express things. However, in wide-coverage parsing, this is a serious limitation. The first experiment in this direction was the conversion of the Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993) to GF RGL. It showed that over 90% of the nodes could be interpreted as RGL functions, but only 5% of the sentences had complete translations (Angelov 2011). This led to a series of extensions as described in Section 4.5, first meant to cover the missing English structures. However, if the ultimate goal is to build an interlingual grammar, the structures designed for English are not necessarily adequate for other languages—in particular, they might not allow for compositional linearizations. Extending the RGL is therefore a slow process by design, and different languages proceed at different speeds.

One example of variation is **word order**. English has an exceptionally rigid word order, but even the Penn Treebank presented variation in the order of complements and adjuncts not previously covered in the RGL. The situation is much more serious in languages like Finnish. Word order is often free only from the weak generative point of view (what strings are grammatical). From the strong generative perspective, Finnish word order often expresses different structures, for instance definiteness (as Finnish has no articles) and focus (Karttunen and Kay 1985). Such structures should be covered by distinct abstract syntax functions, which in turn should be implemented for other languages. Just some small experiments have been made in this direction (Ranta 2012).

The abstract syntax of RGL can sometimes look too superficial even for syntax-based translation purposes. One example is the **tense system**. There is a common abstract tense system, which works well for English and Scandinavian languages, but is insufficient for many other languages. For instance, Romance languages make an aspectual distinction between continuous and "simple" past. This is implemented in the Romance extensions to RGL, but there is no support for selecting the right tense when translating English to French: this has to be done on an additional semantic level.

A GF-inherent problem, rather than an RGL-specific one, is that the need to implement a common abstract syntax can require complex record and table systems, which leads to heavy run-time grammars (cf. Section 3.6). For instance, the slash constructions (Section 4.3) in Romanian become so heavy that the grammar cannot be used for parsing in its entirety, but only as a library.

A general consequence of a common abstract syntax is that spurious parsing ambiguities result in almost all individual languages. For instance, indefinite and definite nouns are mostly indistinguishable in Finnish and Slavic languages, which means that a noun used as an NP typically gets two parse trees. The same happens with the tense system, where for instance Finnish does not distinguish between the present and future of verbs. A slightly more subtle ambiguity can be found in predicative constructions involving interrogatives:

Who is the chairman?

has two analyses: one where *who* is the subject and another where it is the complement. The resulting two trees have different linearizations in some languages, for example, Finnish (*kuka on puhemies* vs. *kuka puhemies on*). And indeed, the ambiguity is not spurious even in English, as shown by the subordinate forms of the questions (*I don't know who the chairman is* vs. *I don't know who is the chairman*).

In some cases, the common abstract syntax has turned out impossible to implement for the reason that a language just does not have that construct. The clearest example is Japanese, where the conjunction of relative clauses (included in the RGL) is possible only for clauses where the relative pronoun is the subject. The grammar does linearize trees for other clauses as well, but they are not grammatically correct (Zimina 2012). One possible solution is a layer of non-compositional post processing that finds paraphrases for inadequate trees (Section 3.5).

Similar reasons have led to a decision not to include a V2 conjunction (as in *John loves and hates Mary*) in the RGL. The problem is that, in most languages, V2 conjunction is only possible for verbs whose complement cases match. But the RGL does not indicate complement case in the abstract syntax (Section 3.4.4).

5. Levels of Interlingua

Abstract syntax can formalize structures on different levels. The highest level conceived by Vauquois, *semantic interlingua*, is a full representation of meaning. In GF terms this means that if we find the correct tree matching the source language, then we can be sure that its linearization into the target language renders the meaning correctly (given that the linearization rules are correct).[21] The next level down is a *syntactic interlingua* such as the RGL, but it gives no guarantees of correctness: A translation can be syntactically equivalent to the source sentence without being a correct expression of its meaning. At the lowest level, one can construct a *lexical interlingua*, where translation is performed by replacing words with their equivalents. Between syntactic and lexical, a *chunk interlingua* works with segments that can be longer than single words. A chunk interlingua can be used to implement a system similar to Apertium (Forcada et al. 2011). It can even take into account morphology—for instance, translate plural genitive nouns into plural genitives. It does not guarantee syntactic correctness, but it can still give good results for closely related languages, such as Spanish-Catalan in Apertium, where word order, morphology, and agreement are similar.

In GF, interlinguas of different levels have been used in two different ways. The most common way, used in numerous CNL systems, is **grammar composition**: interlinguas of lower levels are used as libraries for implementing interlinguas of higher levels (Section 5.1). A more experimental way, used in wide-coverage translation, is to use **layered interlinguas**, where lower levels serve as backups of higher levels (Section 5.2).

5.1 Grammar Composition

The idea of grammar composition is related to NLG pipelines such as the Meaning-Text Theory of Mel'cuk (Mel'cuk 1997). Instead of writing linearization rules directly from a semantic interlingua to strings, tables, and records, we write them in terms of RGL functions that build syntactic structures.

Let us consider a simple example: A social media application might need a semantic predicate expressing that "someone likes something." The abstract syntax function uses semantic categories:

21 Semantics is here understood in the wide sense of including various aspects of *pragmatics*, i.e., encoding all relevant information about a speech act in a relevant situation. Ranta, Détrez, and Enache (2012) is an example of semantics in GF in such a wide sense.

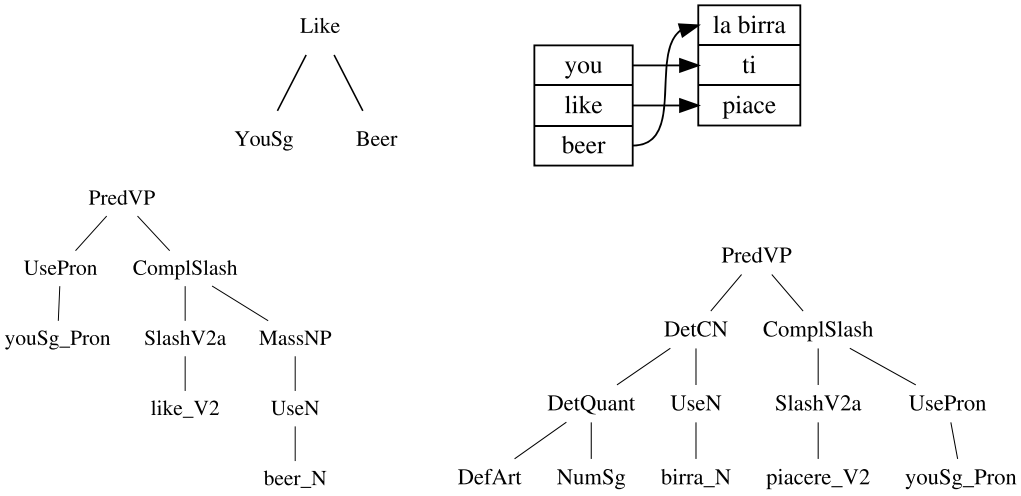


Figure 10
Semantic interlingua and word-alignment (top), followed by syntactic interlingua (bottom), for English and Italian.

```
fun Like : Person -> Object -> Fact
```

The categories `Person` and `Object` are both implemented as NP in the RGL, whereas `Fact` is implemented as clause (Cl).[22] Distinguishing the types `Person` and `Object`, even though syntactically the same, imposes a semantic control of well-formedness: We can express the fact *John likes beer* but exclude *beer likes John*. But because of the use of `NP` and `Cl`, we can use RGL functions to implement the predicate in different languages:[23]

```
lin Like x y = mkCl x (mkV2 like_V accusative) y  -- Eng
lin Like x y = mkCl x (mkV2 gilla_V accusative) y -- Swe
lin Like x y = mkCl x (mkV2 tykätä_V elative) y   -- Fin
lin Like x y = mkCl y (mkV2 piacere_V dative) x   -- Ita
```

In English and Swedish, the construction with direct transitive verb (V2 with the accusative case) is used, with appropriate verbs from the lexicon of each language. In Finnish, the verb *tykätä* takes its complement in the elative case (the "from inside" case), rather than as a direct object. In Italian, the verb *piacere* takes a dative object (either a clitic pronoun or a noun phrase with the preposition *a*), but its syntactic dative object expresses the logical "subject," and the syntactic subject expresses the logical "object."

The effect of the levels of interlingua can be seen by comparing the syntax trees and the word alignment. Figure 10 shows the shared semantic tree as well as the language-specific syntactic trees for English and Italian. It also shows the word alignment constructed

22 "Implemented" means that semantic categories use RGL categories as linearization types.
23 The definitions use the RGL API, where almost every function forming a tree of type C has the overloaded name `mkC`, and word lemmas have names of the form word_C.

by linking words that correspond to the same subtrees in the abstract syntax: Here, the components *you*, *like*, and *beer*. In addition to the subject–object swap, Italian differs from English by adding the definite article to the mass noun *birra* when used as a subject. A grammar using the RGL as a library resembles an NLG pipeline that goes via different levels of representation: semantic, syntactic, and lexical. However, in typical GF systems, the intermediate representations are not visible at run time: All that remains are the semantic trees and the concrete strings. Making this possible in the grammar compiler was a major challenge in the development of GF, using compilation techniques such as partial evaluation (Ranta 2004b; Angelov, Bringert, and Ranta 2009).

GF grammar composition is analogous to transducer composition in the finite-state world (Kaplan and Kay 1994), where grammar writers can structure their source code into smaller components, but this structure is optimized away from the run-time system, which works as a single end-to-end transducer. Borrowing the notation of XFST, we could express the implementation of a semantic grammar for a language, say Hindi, as

```
SemanticAbstract .o. SemanticHin .o. SyntacticHin
```

While grammar composition is an appealing idea linguistically, it also has the practical advantage that it enables a *division of labor* between linguists, who write syntactic grammars, and domain experts, who write semantic grammars. The linguists typically do not know the semantic structures in different domains, which can be very specialized, whereas domain experts do not know the concrete details of languages.

Grammar composition is not restricted to a fixed number of levels, but can exploit any intermediate levels of abstractions, such as the FrameNet, which lies between the RGL API and the final application, as described in Section 7.2.

5.2 Layered Translation

A semantic grammar can only deal with expressions that are interpretable in the semantic model. It can therefore be expected to be partial. To change this behavior, one can use a syntactic grammar as a backup for everything that cannot be interpreted in the semantics. The syntactic grammar can in turn be backed up by a chunk grammar. In general, a grammar can consist of layers of less and less precise grammars, giving fewer and fewer guarantees of translation equivalence, but with the advantage of being robust in the sense of always returning something.

Four layers have been used in Angelov, Bringert, and Ranta (2014), Ranta (2014), and Ranta, Unger, and Vidal Hussey (2015):

- semantic interlingua (a CNL abstract syntax),
- syntactic interlingua (the RGL abstract syntax),
- chunk interlingua (sequences of RGL phrases), and
- verbatim string literals.

Semantic and syntactic interlinguas have been discussed before, whereas verbatim strings need little explanation: An unknown word in the source language is returned as itself in the target language. But what is the chunk interlingua?

Just like the semantic interlingua, a chunk interlingua can be implemented in terms of the RGL. It consists of an extension of the RGL with two new categories: chunks and

lists of chunks. Chunks can be built from different phrase categories, such as NP, AP, VP. The upper limit is an entire sentence, the lower limit a single word. The parser tries to build chunks of maximal lengths. Its output is a list of chunks, which are independent of each other. Linearization renders the chunks into their translations in the same order.

The main difference between full syntactic trees and chunk lists is that chunk lists do not impose any dependencies between the phrases—in particular, no agreement, and no syntactic clue that could change the order of chunks. Thus a chunking parser can recover from agreement errors and return a target sentence, which often has similar agreement errors (here translating from English to Italian):

he sleep here ⟶ `[he_NP, sleep_V, here_Adv]` ⟶ *lui dormire qui*

The middle term is a list of three chunks belonging to different categories.

Inside each chunk, agreement works. This means that longer chunks are better than shorter ones, provided that the chunk limits are correct. The following examples show how both agreement and word order get corrected when chunks get longer:

this yellow house
⟶ `[this_Det, yellow_A, house_N]` ⟶ *questo giallo casa*
⟶ `[this_Det, (mkCN yellow_A house_N)]` ⟶ *questo casa gialla*
⟶ `[mkNP (this_Det (mkCN yellow_A house_N))]` ⟶ *questa casa gialla*

This effect is similar to **smoothing** in n-gram based language models, where longer n-grams are preferred but can be backed up by shorter ones. In grammar-based approaches, GF chunks resemble the **fragments** in lexical functional grammar (LFG) (Riezler et al. 2002), with the expected difference that chunks in GF have a language-independent abstract syntax.

5.3 Evaluations of Layered Translation

GF systems built for CNLs usually aim for a precision of 80% or more, when measured as BLEU scores using post-edited machine translations as reference. Post-edited references are laborious to produce, but they solve one problem typical of rule-based MT, which is that it often produces translations that are, although correct, not the same as in a static reference. Static references are also unfair for statistical and neural translations, but do make more sense when the test set and the training set are selected from the same material. Below, when we compare GF translation with other methods, we always work on separate references for all systems in order to guarantee a fair comparison. Even with post-edited reference, 100% is hard to reach, because human post-editors tend to make changes to improve the style or idiomacy of the translation.

A comprehensive CNL evaluation campaign was made within the European MOLTO project (Rautio and Koponen 2013).[24] The results confirmed BLEU scores mostly above 80%—sometimes under this, sometimes above 90%—for the main CNL applications, such as a tourist phrasebook (Ranta, Détrez, and Enache 2012) and a GF implementation of Attempto-controlled language (Kaljurand and Kuhn 2013).

Going down to lower levels of interlingua expectedly lowers the quality of translation. The first comprehensive experiment in this direction was a translation system

24 Multilingual Online Translation, FP-ICT-247914.

Table 6
Quality evaluation for layered translation in Ranta, Unger, and Vidal Hussey (2015). "CNL" means translation with the semantic grammar, "robust" means CNL with other layers. The "correct" percentage indicates that no changes were made in post-editing.

Language	Correct	BLEU, GF	BLEU, Google
Finnish, CNL	48%	**77**	31
Finnish, robust	0%	**31**	20
Finnish, all	46%	**73**	28
German, CNL	44%	**75**	37
German, robust	0%	33	**34**
German, all	42%	**73**	37
Spanish, CNL	34%	**76**	28
Spanish, robust	0%	**39**	25
Spanish, all	32%	**74**	28

for descriptions of buildings from a text database, with Swedish as source and Finnish, German, and Spanish as target languages (Ranta, Unger, and Vidal Hussey 2015). The input was not pure CNL, but free texts written by many different humans, containing typos and often in telegraphic style dropping articles, etc. Table 6 shows BLEU scores above 75 for semantic translations and 30 to 40 for the chunk-based back-ups. Even the latter were better on the average than Google Translate, except for German.[25]

The size of the system in Table 6 is in the order of thousands of concepts. Section 7.1.5 goes up one order of magnitude and reports the most recent results from translation by using the RGL in combination with WordNet for English and Bulgarian. The proportion of necessary chunking and its effect on BLEU is reported later in Table 12.

The next step is a full-scale open domain task. The GF entry in the WMT contest in 2015 was the first attempt in this direction. It showed very low BLEU scores for a generic English–Finnish translation grammar not adapted to the domain (Kolachina and Ranta 2015).[26] Using the training data of the WMT task to build a specialized lexicon for words and multiword constructions would probably have helped. An interesting question for future work is to develop alignment techniques that can use parallel data to extend grammars with constructions usable as translation units.

5.4 Constructions as Translation Units

Many recent GF projects have addressed "semi-wide-coverage" tasks, which involve legacy texts that are from limited domains but not CNL. The largest such project known to us is a lexicon for the concepts in the General Data Protection Regulation (GDPR) of the European Union (EU).[27] It is a commercial project carried out by a language

25 Google translation at the time was still phrase-based and not neural, at least for Swedish and Finnish. It was chosen as comparison because the customer had used it before ordering the GF system. To guarantee fair comparison, Google translations were evaluated against their own post-edited references.

26 The BLEU score for the GF system was 4.8 according to the EuroMatrix evaluation matrix, `http://matrix.statmt.org/`, whereas the best system in the competition achieved 16.0.

27 `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:32016R0679`.

Table 7
Example of the GDPR lexicon, showing a part of the search results for the word *subject*. Every line corresponds to a different abstract syntax construction involving this word.

English	German	French	Italian	Spanish
be subject to X	X unterliegen	être soumise à X	essere sottoposta a X	ser sujeta a X
be subject to X	X unterworfen sein	faire l'objet de X	essere soggiogata a X	ser sujeta a X
subject	betroffen	concerné	soggetto	sujeto
subject	Gegenstand	sujet	soggetto	objeto
subject-matter	Gegenstand	objet	oggetto	objeto
subject to X	vorbehaltlich X	sous reserve de X	soggetto a X	siempre que se den X
subject to X	fallend unter X	soumis à X	soggetto a X	interesados a X

Table 8
Statistics of the GDPR grammar and corpus. The numbers of different types of abstract syntax functions are independent of language. "Pure syntax" means combination rules with no content words inserted. "Constructions" means mixtures of syntax and content words. "Multiword functions" means linearizations that introduce more than one token; one can see clearly that German is a compounding language. "Multiword frequency" gives two figures: how many lexical items in the corpus are multiwords, and how many tokens are parts of multiwords.

	Abstract	English	German	French	Italian	Spanish
functions, total	3,525	–	–	–	–	–
functions, atomic	3,272	–	–	–	–	–
functions, pure syntax	139	–	–	–	–	–
functions, constructions	114	–	–	–	–	–
word tokens	–	55,186	54,903	62,198	55,296	57,383
word types	–	2,625	4,153	3,206	3,520	3,498
lemma types	–	2,555	3,053	2,467	2,689	2,478
multiword functions	–	590	227	574	559	594
multiword frequency %	–	8–13	3–10	10–21	10–20	11–21

technology and a law technology company, but has a publicly available demo.[28] Table 7 shows an output of the demo, with constructions containing the word *subject* in English with their translations to German, French, Italian, and Spanish. Every line corresponds to a different abstract syntax function, where those with the letter X are constructions that take arguments.

All correspondences in the GDPR grammar are attested in the corpus used. They have concrete syntax rules in all languages addressed, which makes it possible to both generate and recognize them in all of their occurrences (inflected forms, discontinuities). Every phrase shown in the Web demo has hyperlinks to the passages where the term occurs. Table 8 gives some statistics of the size of the corpus and the grammar.

The GDPR grammar was built on top of GF-ACE (Ranta and Angelov 2010), a GF implementation and multilingual generalization of Attempto Controlled English (ACE, Fuchs, Kaljurand, and Kuhn 2008). It was designed to be used for generating, analyzing, and translating legal documents in the data protection domain.

28 See `https://gdprlexicon.com` for the demo and more details.

The GDPR corpus shows a large number of multiword constructions that are not translated word to word. A similar observation was made in the WordNet translation experiment (Section 7.1.5) as well as earlier, in a different context, in the hybrid machine translation experiment on patent claims (Enache et al. 2012). The patent translation project developed some techniques to extract grammatical rules for multiwords from parallel corpora via SMT phrase tables. But this method did not work equally well in the GDPR case, probably because of the much smaller amount of data.

6. Abstract Syntax and Universal Dependencies

UD (Nivre et al. 2016) is an approach to dependency parsing that uses the dependency labels and part of speech tags for different languages. In that sense, UD can be seen as an abstract syntax approach. In this section, we will study the relations between GF and UD in some detail. We will start with a more general discussion of the relations between abstract syntax trees and dependency trees: The right level of comparison is not GF–UD, but either RGL–UD or GF–dependency parsing, because RGL is just one grammar in the GF format, in the same way as UD is just one annotation scheme using the dependency format.

6.1 From Abstract Syntax Trees to Dependency Trees

We need to distinguish between three kinds of trees:

- abstract syntax trees, which are built from typed constructor functions,
- phrase structure trees, which are built from categories (non-terminals) and tokens (terminals), and
- dependency trees, which are built from tokens (words) and labeled dependency arcs.

The labels in dependency trees are meant to stand for **grammatical functions**, such as subject, object, and modifier. This is not the same notion of function as abstract syntax functions in GF. But the notions are related: Grammatical functions in dependency trees correspond to argument places of abstract syntax functions. Some such correspondences are shown in Table 5. To repeat an example, the predication function together with its type and dependency labels is

```
fun PredVP : NP -> VP -> Cl -- nsubj head
```

which means that the first argument, an NP, contains the subject, whereas the second argument, a VP, contains the head. In dependency trees, the subject and the head are not phrases like NP and VP, but individual words in them, namely, the ones that are reached by following the head paths (the **spines**) down to the leaves. In a noun phrase the head is typically a noun; in a verb phrase, it is often a verb.[29] The precise algorithm for deriving a dependency tree from an abstract syntax tree is the following:

29 UD deviates from the latter rule by letting content words be heads if the verb is a copula; this is in fact analogous to the way it is done in the RGL, where the copula has no abstract syntax function of its own.

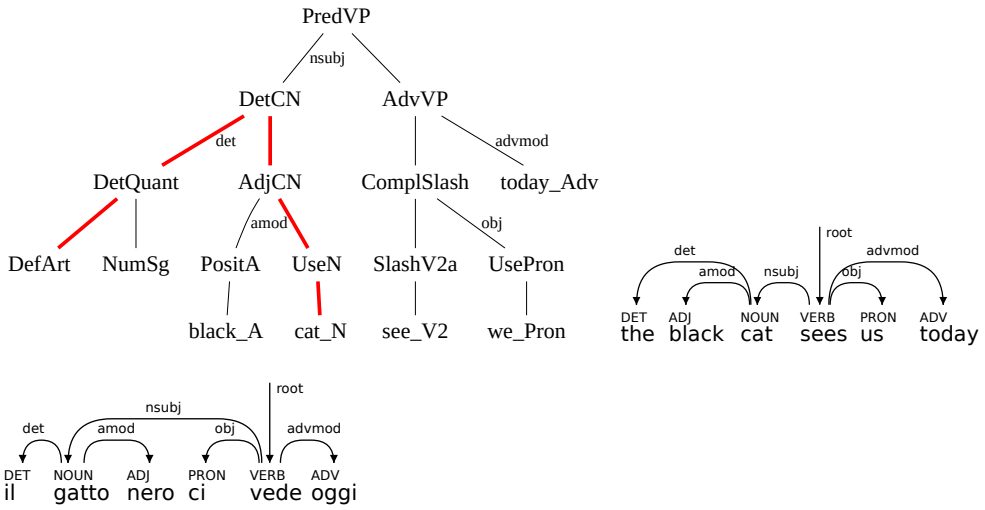


Figure 11
An RGL abstract syntax tree annotated with dependency labels, and the corresponding dependency trees for English and Italian. The thick red path from `DefArt` to `cat_N` shows how the dependency tree algorithm finds the head of a word by walking up the tree until a label is reached, and then walking down along the head spine.

1. Mark the subtrees of each function node with the corresponding labels.
2. Linearize the tree to the desired language, retaining a link from each leaf node to the corresponding word in the linearization.
3. For each word *W*,
   (a) follow the path up the tree until you encounter a label *L*.
   (b) from the function node dominating *L*, follow the spine (head path) down to a word *U*,
   (c) draw an arrow from *U* to *W* labeled by *L*.

This algorithm together with the dependency label annotations for UD is called `gf2ud`. It is the way the dependency trees have been derived in Figure 11.

The algorithm makes an assumption that holds in Figure 11 but not in general: that the leaf nodes in abstract syntax trees are in one-to-one correspondence with words in linearizations. There are three exceptions to this:

- A leaf node has no word attached, as, e.g., in the case of pro-drop.
- A leaf node has many words attached, as in multiword expressions.
- There are words not corresponding to any leaf node.

The first exception is mostly harmless: Some edge is just not created, resulting in the same tree as in standard UD. This can be a problem if the node is a head; however, the UD convention of using content words as heads usually avoids this situation.

The second exception, multiwords, is a problem even inside UD, solved by more or less established ways to impose some internal structure, for instance, on multiword prepositions such as *in accordance with*, *on behalf of*.

The third exception is the most serious one. It has to do with **syncategorematic words**: words that do not belong to categories, but are introduced in connection to syntactic combination rules. In the RGL, this includes copulas, negation words, tense auxiliaries, and infinitive marks. For example, the copula in sentences like *the cat is black* is often missing in Arabic, Russian, and Chinese (depending on tense and some other factors). Therefore it is not considered to be a universal meaning-bearing unit, and it is introduced in the linearization rules of adjectival and other non-verbal predication, for those languages that need it.

The dependency configurations such as given in Table 5 and used in the above algorithm are called **abstract configurations**. They apply to all RGL languages and produce UD labels for all words that correspond to abstract syntax leaf nodes. The rest of the words—the syncategorematic ones—are by default given the label `dep` and linked to the head of the closest dominating phrase node. However, this labeling can be changed by using **concrete configurations**, which are rules referring not only to abstract syntax but also to the words produced in linearization. For example, the following concrete configuration marks the finite forms of the English verb *be* with the label `cop` when used as copula with a predicative complement, links them to the head of the `UseComp` function, and marks them with the part of speech tag `VERB`:

```
UseComp "is","are","am","was","were" VERB cop head
```

With appropriate shorthands and abstractions, a dozen or two concrete configurations are enough to label all words of a language with an informative UD label (i.e., other than the dummy label `dep`). Looking from another perspective, about 80% of text words on the average get correct labels by the abstract configurations alone (Kolachina and Ranta 2016). These statistics become relevant when we consider GF as a tool for generating dependency trees automatically from grammars (Section 6.3).

Abstract and concrete configurations, as described above, are **local** in the sense that they refer only to one function and its immediate subtrees. They could also be called **compositional** in a sense related to Section 3.5. However, if we want to match the standard UD treebanks exactly, local rules are not always enough. This is often due to language-specific variations. For instance, the combination of VP-complement verbs (VV) has two alternative configurations,

```
fun ComplVV : VV -> VP -> VP -- head xcomp | aux head
```

depending on whether the VV is considered auxiliary or main verb. The RGL does not make this distinction, because being auxiliary is not universal in the same way as being a VP-complement verb is. In some languages, such as Finnish, this distinction hardly makes any sense, because there is no corresponding difference in syntactic behavior. In some languages, such as the related Germanic languages English and Swedish, the distribution is different: The translation of *I want to sleep*, where *want* is a head verb, is *jag vill sova*, where *vill* is an auxiliary. In the RGL, auxiliarity is just a parameter in the concrete syntax, and only in those languages where it makes sense.

One possibility is to ignore some details of current UD, and work with dependency trees that are a better match to RGL abstract syntax and, in this technical sense, also

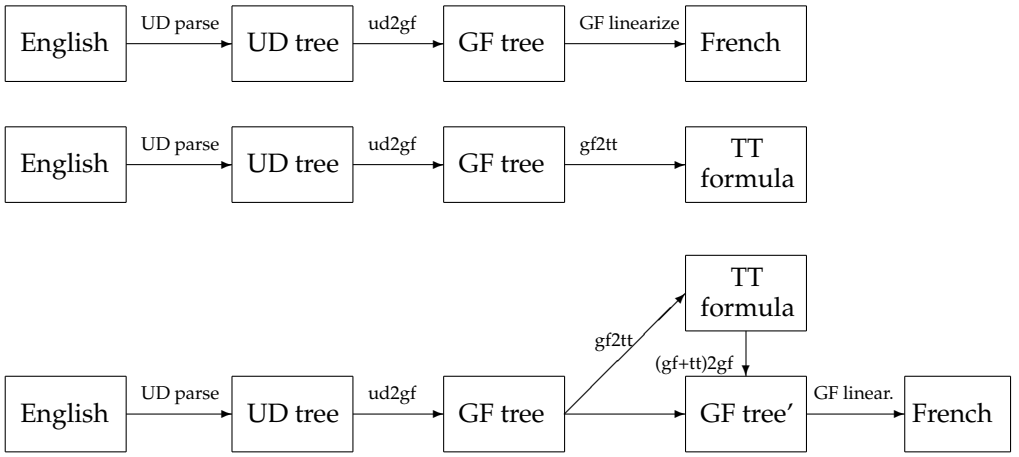


Figure 12
Three pipelines combining UD with GF: direct translation, type-theoretical semantics (gf2tt), translation instructed by semantics.

more universal. Another possibility is to use non-local rules to satisfy the current UD conventions, which means more work specific to individual languages.

6.2 From Dependency Trees to Abstract Syntax

Converting abstract syntax trees to dependency trees is a deterministic procedure that can be carried out by a set of rules. But how is the opposite direction: Can we retrieve GF trees from dependency trees? The utility of such a procedure for GF applications would be high, because parsing in GF can be slow and brittle (Section 3.6), whereas dependency parsing is fast and robust. On the other hand, dependency parsing has no obvious way to generate language—in particular, to convert a tree to a string in a different language. This makes UD trees alone insufficient as a translation interlingua.

Combining dependency parsing with GF generation can be used as a translation pipeline. Figure 12 shows three variants of it. The first one is direct translation: UD parsing piped into GF generation. The second one uses the semantic back end of GF to produce a logical form, here a type-theoretical formula (Section 3.5). The third one uses the logical analysis to guide translation.

Guidance from logical analysis can be used, for instance, in the translation of anaphoric expressions, to guarantee that no ambiguities are created in translation. For example, the English sentences (1) and (2) below are both translated to French (3), if pronouns are compositionally rendered as pronouns. But a logical analysis that first finds the referents of the English pronouns (via bound variables in the logical formula) can replace them by the unambiguous translations (4) and (5), as described in Ranta (1994):

1. *if a man owns a donkey he beats it*
2. *if a man owns a donkey it beats him*
3. *si un homme possède un âne il le bat*

4. *si un homme possède un âne l'homme le bat*
5. *si un homme possède un âne l'âne le bat*

In order to build pipelines like those in Figure 12, we need a `ud2gf` component to retrieve abstract syntax from dependency trees. An algorithm for this is presented in Ranta and Kolachina (2017). Its core is the same kind of dependency configurations as in gf2ud, which are now used in a search mode: Given a UD tree, find all GF trees that can generate it. The ud2gf conversion is not deterministic, since gf2ud is a lossy operation. For instance, the order of adjectival modifiers is not preserved in UD: The French phrase (6) has just one UD tree, whereas the RGL gives two trees corresponding to two different English translations:

6. *petit animal carnivore*
7. (*petit animal*) *carnivore*—"carnivorous small animal"
8. *petit* (*animal carnivore*)—"small carnivorous animal"

Another problem is even more serious in practice: that many existing UD trees have no corresponding RGL trees. This can have many reasons:

- GF grammars do not cover everything.
- UD treebanks have annotation errors.
- UD parsers return wrong parses.

The way to cope with this in Ranta and Kolachina (2017) is to introduce **backup rules**, which interpret unknown parts of UD trees as adverbial modifiers of closest possible subtrees. This makes it possible to build an RGL tree from any UD tree, but some subtrees are interpreted as modifiers with unclear grammatical status. This idea is related to chunk grammars (Section 5.2), but it has better accuracy, since the backup phrases are attached to local subtrees, and their presence does not need to break the overall structure of the sentence. Table 9 gives statistics of the coverage of nodes in terms of how much such backups are needed in parallel UD treebanks. Figure 13 gives an example of a partially interpreted UD tree and the generated GF-RGL tree.

Using gf2ud as a front end to GF generation is an example of **explainable machine translation** (XMT), which uses a neural parser as a black box (Straka and Straková 2017), but returns an **explanation**—the interlingual tree—which makes it possible to assess the reliability of the translation (cf. Figure 5 in Section 2).

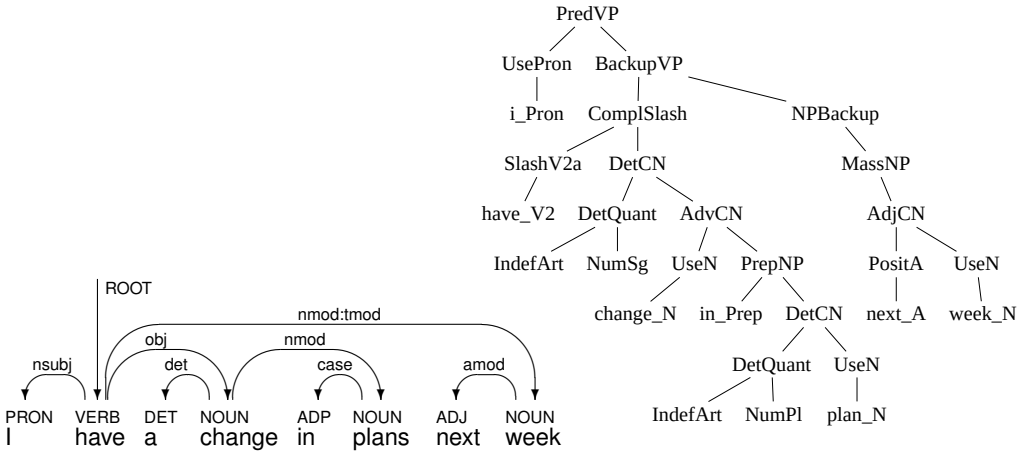
6.3 Data Augmentation and Bootstrapping

We saw in the previous section how dependency parsing can be used as a component in GF pipelines. GF generation can, conversely, be useful in dependency parsing projects. Here are three obvious ways how to use GF to save manual work:

1. **Data augmentation**: Extend a dependency treebank by new trees derived from given trees.
2. **Data transfer**: Translate a treebank from one language to another.

Table 9
Coverage of nodes in test sets from the PUD treebanks for English, Swedish and Finnish. L* (Swedish*, Finnish*) is with language-independent configurations only. #conf's is the number of language-specific configurations.

Language	# Trees	# Confs	% Interpreted trees	% Interpreted nodes
English	1,000	59	77	96
Finnish	1,000	49	69	95
Finnish*	1,000	0	62	84
Swedish	1,000	62	71	94
Swedish*	1,000	0	64	83



I have a change in plans [next week]
minulla on muutos suunnitelmissa [seuraava viikko]
jag har en ändring i planer [nästa vecka]

Figure 13
A tree from the UD English training treebank interpreted in GF with a backup node, and the resulting linearizations to English, Finnish, and Swedish, where the backup part is enclosed in brackets. The grammar used here does not recognize the prepositionless noun phrase as an adverbial. The Swedish translation happens to be acceptable, whereas the Finnish translation would require a different case to make sense.

3. **Data bootstrapping**: Synthesize a treebank from a grammar with no previously given trees.

Data augmentation (Figure 14) uses both ud2gf and gf2ud. The "augment" step is carried out at the level of GF trees. It can be a simple process such as generating all tense and polarity variants of a clause. But it can also vary the subject, the verb, and the object, as well as add or remove modifiers. A handful of simple tree operations can multiply the size of the treebank by several orders of magnitude. Only a part of this makes significant difference when training a parser by machine learning, and there

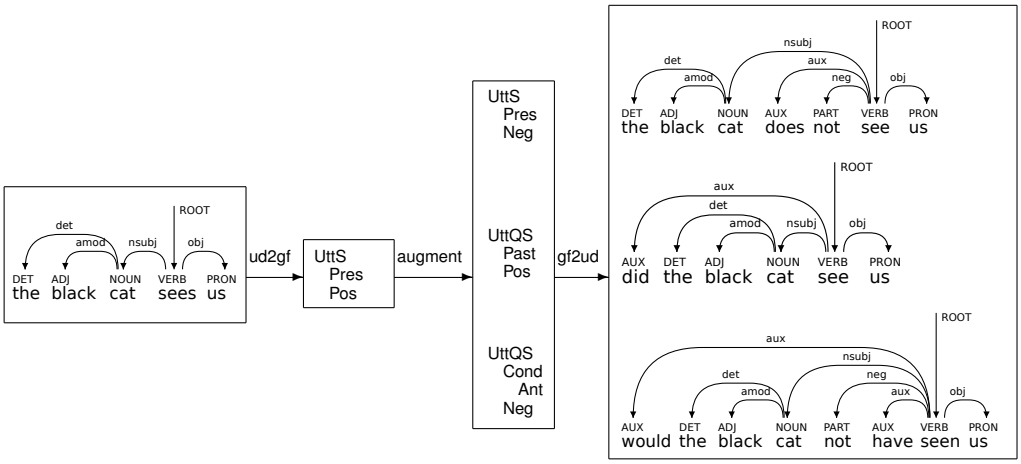


Figure 14
Data augmentation: The tree for a sentence is transformed to trees in other tenses, polarities, and question/declarative forms. In the generated trees, morphological tags (in the UD notation) can be included to improve the training, even though not shown here.

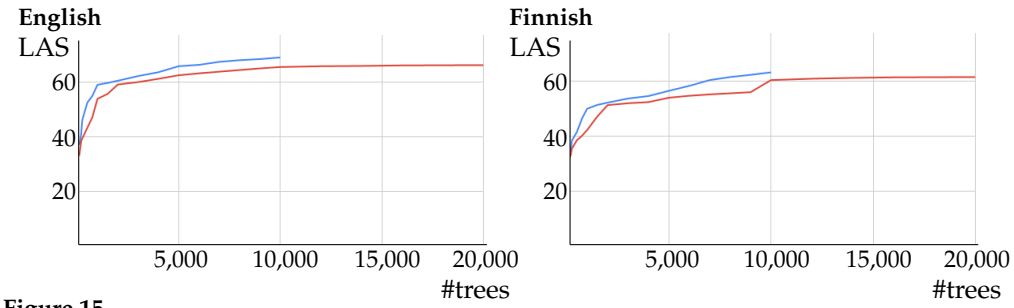


Figure 15
Data bootstrapping: learning curves with synthetic (red) vs. natural (blue) treebanks for English and Finnish, showing the Labeled Attachment Score (LAS) as a function of training corpus size. Models trained on synthetic data need approximately twice the size of the real examples to achieve comparable scores in delexicalized parsing tasks. The results are from Kolachina and Ranta (2019).

are computational limits to how much data the learning can actually cope with. [30] The potential of the method is to help cover all grammatical variations in the language (especially morphologically complex and/or under-resourced ones). Such potential has previously been shown by Sahin and Steedman (2018), whose methods are based on string-level edits of trees; the advantage of grammar-based augmentation is that it can guarantee the well-formedness of the resulting trees. Initial experiments on data augmentation show improvements of 1% to 2% units in the performance of labeled attachment scores (LAS).

Data transfer is related to augmentation, because it uses in a similar way both ud2gf and gf2ud. But the linearization involved in the gf2ud phase targets a language different

30 A naive approach to augmentation increases the size of the treebank by a factor of 1,000.

from the source. The augmentation phase can adapt the data to the target language, for instance, by adding examples of morphological variation not found in the source language. This method is an instance of **crosslingual treebanking**, which in Tiedemann and Agić (2016) is approached by statistical machine translation methods. Just like in data augmentation, one could expect better precision if grammatical knowledge is used in the generation phase. Kolachina and Ranta (2016) give some examples of this, but a large-scale evaluation remains to be done.

Data bootstrapping is the most radical method, of particular interest for languages that have grammars but no treebanks. The method produces a set of trees from a GF grammar and converts them to UD. No ud2gf is needed, but only gf2ud. Figure 15 shows, following Kolachina and Ranta (2019), that 20,000 synthetic trees can be as good as 10,000 manually annotated trees, both in English and Finnish. As the process is fully automatic, no extra work is required to produce any number of trees—millions or billions. However, as noticed in Kolachina and Ranta (2019), the learning curves flatten on a lower accuracy level than manual treebanks, as soon as the variations covered by the grammar are exhausted. An obvious next question is what combinations of bootstrapping, manual annotation, and augmentation are the best ways to build parsers with the minimum of manual work.

All three methods share a general problem of synthesized data: how to guarantee not only that all relevant variation is covered, but also that the distribution of data is realistic. When training a statistical parser, wrong distributions can lead to wrong behavior. Data of natural origin, if sampled in a proper way, is expected not to have this problem (although even there the distribution may vary from one domain or genre to the other).

Interlingual representations such as GF and UD suggest a simple solution: One can transfer a probability distribution of syntactic structure (that is, abstract syntax functions or dependency labels) from one language to another, and expect it to be, if not optimal, much better than a random distribution. In the bootstrapping experiment of Kolachina and Ranta (2019), the distribution from the RGL version of the English Penn Treebank was used. The results showed some drop of performance from English to the other languages, but even Finnish ended up above 60 LAS points just below English.

6.4 The Goals of UD and RGL

To conclude the comparison between UD and the GF-RGL, let us take a look at "Manning's Law," a set of rules followed by the developers of UD.[31] We annotate each rule (in italics) with a comment on how it is followed in the GF-RGL.

1. *UD needs to be satisfactory for analysis of individual languages.* RGL tries to follow this rule as well, moreover with an explicit concrete syntax for each language.
2. *UD needs to be good for linguistic typology.* RGL enables typological comparisons, by sharing structures on different levels.
3. *UD must be suitable for rapid, consistent annotation.* This was not among the goals of RGL, but can be done—for example, by post-processing UD annotations.

31 Not found in publications, but stated, e.g., in `http://universaldependencies.org/introduction.html`.

4. *UD must be suitable for computer parsing with high accuracy.* Parsing with RGL is in principle more accurate in details, but its coverage is worse and its speed lower.
5. *UD must be easily comprehended and used by a non-linguist.* This is the goal we have set to the RGL API, but not to the RGL itself. However, the gf2ud mapping can be used to communicate RGL analyses to those who understand UD.
6. *UD must provide good support for downstream NLP tasks.* Definitely one of the main goals, even though the typical downstream tasks are different from those with UD.

7. An Abstract Syntax View on Other Approaches

In most grammar formalisms used for natural language (HPSG, LFG, TreeAdjoining Grammar [TAG], Combinatory Categorial Grammar [CCG]), abstract syntax is closely coupled with concrete syntax.[32] Thus for instance the F-structures of LFG contain both semantic and morphological features. Grammars are written separately for different languages, even when packages of parallel grammars are maintained (LinGO, ParGram). There is no common abstract syntax or other kind of interlingua.

Some data-driven approaches, such as UD, can be easier to relate to GF, because they typically have abstract representations defined separately from relations to concrete languages; these relations are often established by machine learning techniques rather than grammatical rules. In this section, we will give an overview of some approaches that have established connections with GF.

7.1 WordNet

WordNet (Fellbaum 1998) started as an effort to enumerate the senses of all English words, and to organize those senses into a network of interrelations. Soon after, the effort was applied to other languages. First the EuroWordNet (Bloksma, Diez-Orzas, and Piek 1996) for Dutch, Italian, Spanish, French, German, Czech, and Estonian, and later the BalkaNet (Stamou et al. 2004) for Bulgarian, Czech, Greek, Romanian, Serbian, and Turkish. There are also a multitude of similar projects for other languages. In particular, all publicly available WordNet resources are aggregated by the Open Multilingual WordNet project (Bond and Paik 2012).

What is interesting about WordNet from the GF perspective is that most of these resources for other languages are also linked to the English WordNet. In WordNet, a set of words that can be used interchangeably (synonyms) to express a particular meaning are grouped together. Each of these **synsets** (synonym sets) has a unique identifier in the English WordNet. For example, Princeton WordNet has the following synset:

```
(08094856-n) family, household, house, home, menage
        (a social unit living together)
```

32 See Pollard and Sag (1994) for HPSG, Bresnan (1982) for LFG, Joshi and Schabes (1997) for TAG, and Steedman (2000) for CCG.

When a WordNet for a new language is developed, the developers usually keep a link between the English synset and the one in the new language. Thus for Swedish for example (Borin, Forsberg, and Lönngren 2013; Viberg et al. 2002), one could find the corresponding synset:

```
(08094856-n) hus, hushåll, familj, hem
      (a social unit living together)
```

From the outside it seems as if the WordNet initiative is building an interlingual lexicon, which fits very well with GF's mission to build an interlingual grammar. This was attempted for instance in Virk et al. (2014). There, every synset corresponds to one abstract function and then the function's linearization in each language produces all words in the language as variants.

However, a more detailed analysis shows that this is not ideal. For instance, with the above assumption it is equally likely that a translation of "family" to Swedish could be either of "hus," "hushåll," "hem," or "familj." In reality, this is true only if we are very sure about the right sense used in a particular sentence. On the other hand if we restrict "familj" to be the only translation of "family" then that choice will be valid even if the actual sense was:

```
(08124465-n) family
    ((biology) a taxonomic group containing one or more genera)
```

where candidates like "house," "household," and so forth, would be inappropriate.

Another problem is that very often the entries in one and the same synset might be synonyms but can be used only in very different regimes. For instance, we have:

```
(12654755-n) apple, orchard apple tree, Malus pumila
  (native Eurasian tree widely cultivated in many varieties
        for its firm rounded edible fruits)
```

Here the Latin name "Malus pumila" is used only in botany, and would be inappropriate as an everyday term for an apple tree.

A third problem is that the authors of different WordNets are prolific to different extents. For instance the synset above includes "orchard apple tree" which may or may not be included in the other languages, even if literal translation of the expression might work equally well. In fact, from the point of view of optimality of the lexicon, multiword expressions should be included only if they lead to a non-compositional translation in another language. Despite this, examples like these occur in many languages, but whether or not the equivalents in other languages are included is not consistent.

These examples suggest that translation equivalence is both more fine-grained and a more coarse-grained relation than synset equivalence. For example, the translation equivalence "family-familj" is more coarse because it can be reused for both synsets 08094856-n and 08124465-n. On the other hand, the translation equivalence should be more fine-grained in order to avoid producing "Malus pumila" in Swedish as a response to "apple." At least in the Finnish WordNet (Lindén and Carlson 2010) that equivalence was explicitly preserved because the whole database was created by manual translation.

Abstract	Bulgarian	English	Finnish	Portuguese	Slovenian	Spanish	Swedish
household_N	домакинство	household	kotitalous	casa	gospodinjstvo	casa	Swedish
house_10_N	къща	house	suku	casa	hiša	casa	Swedish
home_8_N	дом	home	perhe	lar	dom	hogar	Swedish
family_1_N	семейство	family	talous	família	družina	familia	Swedish

Figure 16
A snapshot of a synset from the search interface for GF WordNet.

An attempt to automatically recover the equivalence from already existing linked WordNets is presented in Angelov and Lobanov (2016). The method, however, is limited by the quality and the size of the original resource. In general, the algorithm tends to produce very good results when the input data is already clean from mistakes and it covers many synsets. Post validation of the result can still be necessary.

7.1.1 A WordNet in GF. There is an ongoing effort to build an interlingual WordNet for Bulgarian, Catalan, Chinese, Dutch, English, Estonian, Finnish, Italian, Portuguese, Slovenian, Spanish, Swedish, Thai, and Turkish[33] that is consistent with GF by design. Because of the wide range of languages, we decided to focus on a few languages first for which there is in-house expertise: Bulgarian, English, and Swedish, whereas for the other languages the lexicon is bootstrapped using the method in Angelov and Lobanov (2016) with very little post-validation. The exception is Finnish, where we used the translation data in FinnWordNet (Lindén and Carlson 2010). This means that the translation choices for Finnish are mostly correct, but further work is needed for multiword expressions and for correcting verb valencies. In order to compensate for the limited coverage of most WordNet resources, additional data for all languages was extracted from PanLex (Kamholz, Pool, and Colowick 2014).

In parallel with the lexicon, we build a corpus that is annotated with abstract syntax trees where the lexical items in the trees are sense disambiguated. The corpus is intended for training statistical parsing (Angelov and Ljunglöf 2014) and sense disambiguation in GF.

A synset in our WordNet is a matrix as in Figure 16, where the rows of the matrix represent the translation equivalence, and the columns represent the synsets of the different languages. Note that each row also corresponds to a different abstract syntax identifier. It is important to note that a single identifier like `family_1_N`[34] could in principle be reused across synsets. For instance, it can also be used for the synset:

```
((biology) a taxonomic group containing one or more genera)
```

since all the chosen translations also fit in that sense. When identifiers are reused, this reduces the size of the interlingual lexicon, but what is more important is that for translation purposes it becomes enough to perform sense disambiguation only up to the level of abstract identifiers and not the actual WordNet sense. This means that the amount of ambiguity can be reduced to the level needed in translation, while the real sense ambiguity is much higher.

The raw data for the synsets comes from existing resources such as Princeton WordNet for English (Fellbaum 1998), Svenska OrdNät (Viberg et al. 2002) and WordNet-SALDO

33 `https://github.com/GrammaticalFramework/gf-wordnet`.
34 The identifiers are composed of an English lemma plus a GF category. When that is not enough, we add a consecutive number to make the identifier unique.

	nom	gen
sg	house	house's
pl	houses	houses'

		obest	best
nom	sg	hus	huset
	pl	hus	husen
gen	sg	hus	husets
	pl	hus	husens

ед.ч.	нечленувано	къща
	членувано	къщата
	пълен член	къщата
мн.ч.	нечленувано	къщи
	членувано	къщите
звателна форма		къщо
бройна форма		къщи

	yksikkö	monikko
nominatiivi	talo	talot
genetiivi	talon	talojen
partitiivi	taloa	taloja
translatiivi	taloksi	taloksi
...	...	...

Figure 17
Snapshots of morphological tables generated from GF WordNet.

(Borin, Forsberg, and Lönngren 2013) for Swedish, and BulTreeBank WordNet for Bulgarian (Simov and Osenova 2010). The BalkaNet WordNet for Bulgarian (Koeva, S., and Genov 2004) was not used because of its restrictive license. Both the Swedish and the Bulgarian resources are incomplete (28,046 senses in Swedish and 8,936 in Bulgarian), so additional translations are added from the Folkets Lexikon for Swedish (Kann and Hollman 2011) and KBEDict for Bulgarian (Angelov 2014). The initial bootstrapped resources are then subjected to an ongoing validation effort. The validation ensures that the best translation pairs are chosen according to the criteria:

- the two words should share as many senses as possible;
- each of the words should be commonly used as translation of the other;
- preserve morphological derivations as much as possible, that is, if *science* is *vetenskap* then *scientific* is *vetenskaplig*;
- whenever possible, preserve cognates between languages; and
- when the data come from Folkets Lexikon or SA Dictionary, check that the translation matches the actual WordNet sense.

7.1.2 Morpho-syntactics. The original WordNet resources are all on the level of lemmas, whereas, as discussed earlier, in GF we also need to have complete morphological information for all languages. A large morphological lexicon for English and Bulgarian already existed (Angelov 2014); to that we also added the morphology for Swedish from SALDO (Borin and Lönngren 2009). The morphology for Finnish comes from Kotus.[35] Whenever we encounter words that are not listed in any of the existing morphological lexica, then those words are inflected by using the smart paradigms in the RGL (Section 3.4.1). Figure 17 shows inflection tables for English, Swedish, Bulgarian, and Finnish generated automatically from our grammar.

Morphology is not the only missing piece. In WordNet all lexical items are categorized as just nouns, verbs, adjectives, or adverbs, whereas the RGL requires finer subcategorization. The most prominent need of subcategorization is for verbs, which exhibit different valency patterns. The grammar, for instance, needs to know whether a verb is intransitive, transitive, or a prepositional verb. Particle verbs are also treated specifically since in translation the particle is often absorbed as a prefix or suffix in

35 `http://kaino.kotus.fi/sanat/nykysuomi/`, the free morphological dictionary of the Institute for the Languages of Finland.

the main verb, and thus, the phenomenon is treated in the concrete rather than in the abstract syntax.

Another example is the adverbs, which in WordNet are treated uniformly, whereas in the resource grammars they are categorized according to their syntactic functions. For instance adverbs like "very," which modify adjectival phrases, have different categories than adverbs like, for example, "occasionally," which typically modify verbs. There are also adverbs like "instead," which in one sense can be used on its own whereas in another is followed by "of" to form the composite preposition "instead of." In the latter case the composite phrase has the category of a preposition in the grammar whereas in the original WordNet it is treated as an adverb. Phrases like "instead of" are treated as multiword expressions with non-compositional translation to other languages.

A third apparent example are numerals, which in WordNet are classified as both nouns and adjectives according to their uses. In the RGL, there is a special category of numerals, which can be used syntactically on their own (similar to nouns), as determiners (the cardinals), or like adjectives (the ordinals).

Nouns are categorized separately depending on whether they are common nouns or proper names.

In all these cases, the categorizations in GF WordNet differ from the categorization in the ordinary WordNet. There is also an ongoing integration with VerbNet (Schuler 2005) that will make the verb subcategorization more stable. The current subcategorization is based mostly on the uses of the verbs in the corpus of examples coming with WordNet.

7.1.3 Examples Corpus. In WordNet, most of the synset glosses contain examples in English showing typical uses of the particular senses. We extracted the examples, and each example was associated with the word that it exemplifies. This corpus is then parsed with GF and sense-annotated. Note that there is the Princeton WordNet Gloss Corpus, which also contains sense annotations but only for some words—in order to construct complete trees we had to annotate all senses.

Once we have an abstract tree for a particular English sentence, we can in principle translate it to all languages for which the grammar is available. However, because the lexicon is still under development, we cannot trust the translations. There are still a lot of mistakes and gaps. To facilitate the development, we have also bootstrapped the translations to Bulgarian and Swedish by using Google Translate. This gives us a corpus of entries like this:

```
abs: DetCN (DetQuant IndefArt NumSg)
           (AdjCN (PositA rare_1_A) (UseN word_1_N))
eng: a rare word
swe: ett sällsynt ord
bul: rjadka duma
key: 1 rare_1_A 00490548-a
```

Each abstract syntax tree is followed by its verbalizations in English, Swedish, and Bulgarian. The English sentence is the original example, and for Swedish and Bulgarian we have either the grammar generated sentence if the entry is already validated, or a Google translation if the entry is still pending.

Bulgarian	рядка дума
English	a rare word
Finnish	harvinainen sana
Portuguese	uma palavra pouco frequente
Slovenian	redka beseda
Spanish	una palabra poco frecuente
Swedish	ett sällsynt ord
word_1_N: a unit of language that native speakers can identify	

Figure 18
Snapshots of an example from GF WordNet. The highlighting shows the word alignment and at the bottom is shown the gloss for the selected word.

Table 10
State of GF WordNet in terms of number of senses and number of trusted translations.

Language	Senses	Trusted
Abstract	107,955	—
Bulgarian	81,728	35,906
Catalan	95,728	4,470
Chinese	41,412	93
English	107,826	107,826
Finnish	97,059	90,535
Portuguese	94,905	53,324
Slovenian	86,489	52,017
Spanish	98,255	5,066
Swedish	76,327	27,998
Thai	64,671	109
Turkish	88,954	3,611

From a user interface (Figure 18),[36] it is possible to find the already verified corpus entries. Because the sentences are generated from a single abstract tree, it is possible to automatically compute the word and phrase alignments. In the user interface, when the user clicks any of the words in a sentence for one language, the corresponding words (phrases) in the other languages are highlighted as well. If the current selection covers only a single word, the user interface shows the gloss for that word below the corpus example.

7.1.4 Current Status. The status of the WordNet lexicon in September 2019 is shown in Table 10. As can be seen, the only complete language is English; this is because the abstract syntax was constructed from English. The next almost-complete language is Finnish, since the original data set had very good coverage. We are actively working on Bulgarian and Swedish, and the coverage for our lexicon is already better than in the original sources. All other languages are automatically aligned; for those languages we marked as trusted only those entries that we spotted and for which we are very sure. Because of the bigger sizes of the original Portuguese and Slovenian resources, the alignment proved to be consistently good. For that reason, entries coming from

36 `https://github.com/GrammaticalFramework/gf-wordnet`.

Table 11
BLEU scores for translation with RGL+WordNet versus Google Translate. RGL reference: reference obtained by post-editing RGL output. Google reference: reference obtained by post-editing Google output. RGL: output produced from manually disambiguated trees in the treebank by RGL linearization. RGL+parse: output produced by parsing the trees with automatic disambiguation and linearizing with RGL.

	RGL reference			Google reference		
	RGL	RGL+parse	Google	RGL	RGL+parse	Google
Bulgarian	**92.03**	45.27	35.74	37.58	22.19	**84.61**
Swedish	**72.69**	40.74	52.88	**65.49**	38.86	65.04

Table 12
Coverage for English to Bulgarian and translation quality comparison between complete syntactic parses and chunks.

Validated sentences:	4,375	(11%)
Parsed sentences:	29,376	(76%)
All sentences:	38,592	
BLEU score for syntactic parses:		45.27
BLEU score for chunks:		27.20

WordNet and not from PanLex for Portuguese and Slovenian are currently marked as trusted.

The corpus of examples contains 38,592 sentences, of which 11% have been verified (see Table 12). The verification ensures that the right abstract syntax tree is used, which includes a correct choice for the word senses. We also check the translations to Swedish and Bulgarian. Many of the translations sound native, but sometimes when English idioms are used in the source, the translation is too literal. This leads to phrases that are syntactically correct and understandable, but unidiomatic. As mentioned in Section 4.4, idioms and special constructions can be handled in semantic grammars, or by adding additional functions to the RGL. However, we limit ourselves to the lexical level in the WordNet project.

7.1.5 Translation Experiments. Although work on the WordNet lexicon is still ongoing, we did preliminary experiments to see how good RGL+WordNet are for the translation of general text. We randomly selected 200 sentences from the section of the examples corpus that is already validated. After that we asked speakers of Swedish and Bulgarian to post-edit the translations that the RGL generates. From the automatic translation and the post-edited output we computed the BLEU scores (Papineni et al. 2002), which can be seen in Table 11.

The difference between "RGL" and "RGL+parse" scores suggests that the greatest impact on GF translation quality comes from disambiguation. The drop is significant but not surprising. The current disambiguation model in GF for both the syntax and the word senses is context-free. This means that the ranking of the possible syntax trees is not as good as it could be. Similarly, instead of real contextual sense disambiguation, for every word we will just pick the most probable sense.

A contextual ranking and disambiguation based on distributional representation models is under development. A pilot experiment is presented in the master's thesis of Kalldal and Ludvigsson (2017). Agirre et al. (2009) show that best results are achieved by using WordNet for supervising the learning of distributional models from large corpora. This is what we plan as well, except that in our case the vector space embedding must be for abstract syntax functions instead of concrete words.

Another question is how the limited coverage of the RGL affects the translation quality. The examples corpus contains 38,592 sentences (Table 12), but of those only 76% can be fully parsed. The rest can be analyzed only by using chunks as explained in Section 5.2 and as applied in Angelov, Bringert, and Ranta (2014). Looking at the examples, it seems as though at least some of the chunking was caused by missing or wrong verb valency subcategorization in our lexicon. Integration with VerbNet (Schuler 2005) could potentially improve the results.

In any case, we want to be able to produce output even for malformed inputs. This means that integration of some kind of chunking is indispensable. To see the effect of that on the translation we selected 50 chunked sentences, post-edited the RGL translation, and computed the BLEU score. As can be seen in Table 12, the result is much lower. Part of the reason is that sometimes even the analysis of the individual chunks might be the wrong one. This happens with or without chunking and is just the effect of our context-free disambiguation. With chunks, however, the effect is even worse because the agreement between different chunks gets lost.

7.2 FrameNet

The Berkeley FrameNet (BFN) (Fillmore, Johnson, and Petruck 2003; Fillmore and Baker 2009) is a lexico-semantic resource based on the theory of frame semantics (Fillmore 1985). An abstract semantic frame represents a cognitive scenario. A set of core frame elements (FE) represents the conceptually required components of the frame and, thus, uniquely characterizes the frame. Core FEs syntactically tend to correspond to verb arguments. Non-core FEs are not specific to the frame and typically are adjuncts. In a sentence, a frame is evoked by a target word which corresponds to a lexical unit (LU)—a pairing of the word and the frame. An LU defines a word sense with respect to the frame and, based on FrameNet-annotated corpus evidence, carries semantic and syntactic valence information about the possible realization of FEs given the LU.

Following the BFN approach, there are framenets available for many languages, which often reuse the BFN frame inventory (Gilardi and Baker 2018). Exploitation of FrameNet-like resources has been appealing not only for linguistic studies but also for a range of advanced natural language understanding and generation applications.

Although the more than 1,000 BFN frames are, to a large extent, language-independent (Gilardi and Baker 2018) and can be seen as a robust semantic interlingua, LUs and their valence patterns are language-specific. Moreover, when it comes to the syntactic patterns, different framenets use different grammar models, categories, and functions. This is where the GF abstract syntax as a FrameNet-complementary interlingua comes into the picture, particularly regarding the multilingual NLG perspective.

A methodological approach for the extraction and generation of a computational FrameNet-based GF grammar and lexicon has been proposed and tested by Gruzitis and Dannélls (2017). The approach leverages FrameNet-annotated corpora to automatically extract a set of crosslingual semantico-syntactic valence patterns (Dannélls and Gruzitis 2014). The language-independent layer of FrameNet—the abstract frames and their semantic valence—is defined in the abstract syntax of the GF FrameNet grammar

```
mkCl                          -- mkCl  : NP -> VP -> Cl
  person
  (mkVP                       -- mkVP  : VP -> Adv -> VP
    (mkVP live_V)             -- mkVP  : V -> VP
    (mkAdv in_Prep place))    -- mkAdv : Prep -> NP -> Adv

Residence                     -- Residence : Resident -> Location -> V -> Cl
    person                    -- Resident (NP)
    (mkAdv in_Prep place)     -- Location (Adv)
    live_V                    -- lexical unit (V)
```

Figure 19
FrameNet functions as an abstraction layer above the syntax-oriented RGL API.

library, while the language-specific layer—the surface realization of LUs and their syntactic valence—is defined in concrete syntaxes (one per language). The abstract syntax of GF RGL, in turn, is used for generalizing and unifying the grammatical categories and functions used in different framenets.

The resulting GF FrameNet grammar library has multiple effects. First, it provides a frame-semantic abstraction layer and API[37] over the syntactic abstraction layer and API[38] provided by GF RGL. Thereby, application grammar developers can primarily manipulate with plain semantic constructors in combination with some simple syntactic constructors for FE fillers, instead of comparatively complex and recursive syntactic constructors for building verb phrases and gluing the clauses together. The GF FrameNet grammar library also handles the language-specific order and agreement of FEs in the generated sentence, based on dominant patterns found in the respective FrameNet corpus. Also note that, from the GF application grammar development perspective, the GF frame constructors can often be specified for all languages at once: in the shared concrete syntax (functor) of a GF application grammar, which even more facilitates porting application grammars to other languages. Second, it provides a high-level means for both relatively wide-coverage NLG and multilingual NLG. Third, it can be used as a unified method for comparing and mapping semantic and syntactic valence patterns and lexical units across framenets of different languages.

Figure 19 illustrates the use of the GF FrameNet grammar library in comparison to the use of the standard GF RGL. Note that the application of the semantic constructor `Residence` is flatter with respect to its arguments in comparison to the syntactic constructors `mkCl` and `mkVP`.

The `Residence` constructor is one of 869 shared syntactico-semantic valence patterns extracted from the BFN and Swedish FrameNet (SweFN) corpora (Gruzitis and Dannélls 2017). These patterns cover 483 semantic frames. For both BFN and SweFN, the concise set of 869 patterns covers 77.5% of the human-annotated examples (out of 57,615 BFN examples and 3,348 SweFN examples). This indicates that the set of shared constructors includes the most frequently used frames and their valence patterns despite the modest amount of the annotated example sentences in SweFN.

There has been previous work on FrameNet-based NLG, which typically focuses on narrow domains or specific applications, and is usually monolingual. For instance,

37 `http://remu.grammaticalframework.org/framenet/`.
38 `http://www.grammaticalframework.org/lib/doc/synopsis/`.

Roth and Frank (2009) have developed an application that generates walking directions based on a set of rules that map route segments to certain semantic frames, and landmarks within the segments to frame elements. Another example is template-based verbalization of certain facts stored in a knowledge base that has been populated by a domain-specific FrameNet-based information extraction system (Barzdins 2014).

7.3 Construction Grammar

Abstract syntax functions represent grammatical constructions. Basic syntactic structures such as NP–VP predication are examples of such constructions, on one end of the scale; word lemmas linked to their inflection tables and other grammatical properties are on the other end. Frames, in the sense of FrameNet, can be seen as a layer above word senses, whereas what is more specifically called constructions are mixtures of combination rules and words, which have special, non-compositional meanings (Goldberg 2013). The core GF RGL has only a few such constructions, with existentials (Section 4.4) as a typical example. But semantic grammars are full of them, to capture all kinds of idioms that could not be translated compositionally without them. As we saw in Section 5.2, they can in a wide-coverage setting be seen as a layer on top of the syntactic grammar, as fragments of a semantic interlingua. They also have the advantage of making complicated run-time transfer unnecessary and thus maintaining the interlingual paradigm.

A study of a standard construction grammar in GF was carried out in Gruzitis et al. (2015) and Borin, Dannélls, and Grūzītis (2018). It addressed the Swedish Constructicon (SweCcn) (Bäckström, Lyngfelt, and Sköldberg 2014), converting the semi-formal descriptions to GF grammar rules in a semi-automatic way. SweCcn specifies more than 400 constructions, from which a GF construction grammar (an extension to the Swedish resource grammar) can be automatically generated for nearly 80% of the constructions, requiring post-editing for the remaining 20%. Initial evaluation of the automatically acquired GF grammar on different test sets of construction examples shows that the achieved recall is already 70% to 90% (Gruzitis et al. 2015). Some of this work has continued on GF code repositories, and it is seen as a major way to improve wide-coverage parsing, which of course is an open-ended task.

Development of constructicons is actively ongoing for many languages (Lyngfelt 2018). The Russian Constructicon, for instance, which currently specifies nearly 700 constructions, is being built in parallel with SweCcn (Janda et al. 2018), reusing the technical infrastructure and the data format of SweCcn. Thus, the SweCcn-in-GF approach can be applied to the Russian Constructicon in a relatively straightforward manner. Moreover, this would open a further research direction on developing a common GF abstract syntax for a multilingual computational constructicon, building on the recent linguistic research in multilingual constructicography by Lyngfelt et al. (2018).

7.4 Abstract Meaning Representation

AMR (Banarescu et al. 2013) has been proposed as a robust whole-sentence semantic representation, and it has gained considerable attention in the research community. Although the original work on both AMR as a representation and AMR as a semantically annotated text corpus (sembank) has been focused on English, there have been promising efforts in pushing it toward a semantic interlingua (Xue et al. 2014; Damonte and Cohen 2018).

Following a similar task in text-to-AMR parsing (May 2016), a recent shared task at SemEval 2017 unveiled the state-of-the-art in AMR-to-text generation (May and Priyadarshi 2017). According to the SemEval 2017 Task 9 evaluation, the convincingly best-performing AMR-to-text generation system (Gruzitis, Gosko, and Barzdins 2017), among the contestants, combines a GF-based generator with the JAMR generator (Flanigan et al. 2016), achieving the Trueskill score (human evaluation) of 1.03–1.07 and the BLEU score (automatic evaluation) of 18.82 (May and Priyadarshi 2017). The JAMR generator alone placed second, achieving the Trueskill score of 0.82–0.85 and the BLEU score of 19.01 (May and Priyadarshi 2017). In both cases, the common approach is to convert the given AMR graph into a tree, and to generate a surface realization from the tree. While Flanigan et al. (2016) transform AMR graphs into spanning trees, and decode the trees using a weighted tree-to-string transducer and an *n*-gram language model, Gruzitis, Gosko, and Barzdins (2017) transform AMR graphs into GF ASTs, leaving the linearization of the acquired ASTs to the GF RGL, which resolves the word order and syntactic agreement, handles function words, and so on.

Note that the combined GF/JAMR system achieved a lower BLEU score than the JAMR system alone. The BLEU score was not binding in the SemEval 2017 Task 9 evaluation. In fact, this output confirms the inadequacy of using the BLEU score in this kind of NLG task where text is generated from meaning representations—sentences generated via ASTs are, in general, syntactic and even semantic paraphrases, if compared to the source sentences of the AMRs. Also note that the approximately 200 handcrafted rules (within the given timeframe) for AMR-to-AST conversion fully covered 12.3% of the evaluation AMRs and partially covered another 36% of the evaluation AMRs. In the combined GF/JAMR system, the incomplete ASTs and, thus, the incomplete linearizations were replaced by linearizations acquired by the JAMR generator. By adding more and more AMR-to-AST conversion rules and increasing the proportion of full GF-generated linearizations, the BLEU score would decrease even more.

Figure 20 illustrates the difference between the GF-based AMR-to-text generator (Gruzitis, Gosko, and Barzdins 2017) and the JAMR generator (Flanigan et al. 2016), given a sample input AMR that the current AMR-to-AST conversion rule set fully covers. Note that the reference is not the original sentence from which the AMR was specified by a human annotator; it is an informed human translation by the authors.[39] Although the fluency of the GF-generated linearization can be argued, it is obviously more adequate, both syntactically and semantically—and, what is important: the GF-generated output is predictable.

It should be noted that the current AMR specification[40] does not represent grammatical tense, plural, and articles. Because of this information loss, one cannot expect that such details will be correctly deduced or guessed, especially without a wider context. Therefore the default linearization by the GF generator (and the guessed linearization by the JAMR generator) of the AMR concept `person` in Figure 20 is in the singular form. Similarly, information about the plural "boys" is lost in the AMR representation in Figure 21 and is not transferred to the output linearization. Regarding the `ARG2` argument of the `convict-01` predicate, this however requires adding corpus-based valencies to the AMR-to-AST transformation.

39 This sample AMR graph is taken from the SemEval 2017 Task 9 evaluation data set, and the original reference sentences are not available to the authors.

40 `https://github.com/amrisi/amr-guidelines/blob/master/amr.md` (v1.2.6).

```
(c3 / convict-01
  :ARG1 (p / person
    :mod (c4 / country :wiki "Iraq" :name (n3 / name :op1 "Iraq")))
  :ARG2 (s / smuggle-01
    :ARG0 p
    :ARG1 (p2 / person)
    :ARG3 (c / country :wiki "Australia" :name (n / name :op1 "Australia"))))
```

JAMR: *Iraq person has been convicted of the person is being smuggled into Australia*
GF: *a person from Iraq is convicted to smuggle a person to Australia*
Human: *an Iraqi person is convicted for smuggling persons to Australia*

Figure 20
The difference between GF-based and JAMR linearization of a sample AMR. The reference is an informed human translation from the given AMR.

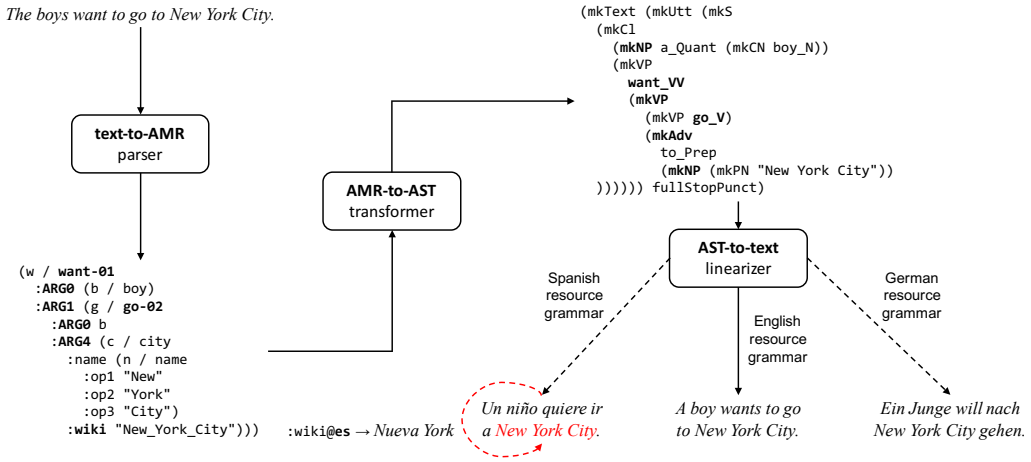


Figure 21
The overall architecture for multilingual AMR-to-text generation via GF.

The alternative AMR-to-text approaches focus solely on AMR-to-English generation. Because the RGL supports many languages, the GF-based approach can be extended to multilingual AMR-to-text generation, given a wide-coverage GF translation lexicon such as the WordNet-based GF dictionary described in Section 7.1. Thus, AMR equipped with GF strengthens the status of AMR being an interlingua.

Figure 21 outlines the architecture of the proposed approach. It also illustrates the requirement and opportunity for a multilingual GF lexicon of named entities—the wikification part of AMR.

In order to develop fully fledged multilingual linearization of AMR graphs via GF, a multilingual Wikipedia-based[41] GF lexicon has to be created first, in addition to the WordNet-based GF translation dictionary. Because AMR representations specify Wikipedia identifiers of named entities (when possible), translation equivalents can be

41 Technically, it can be based on the DBpedia or Wikidata knowledge base, for instance.

found via the Linguistic Linked Open Data cloud (Chiarcos, Hellmann, and Nordhoff 2012), while contextually correct inflectional forms of named entities can be provided by RGL. Thus, the Spanish linearization in Figure 21 would use "Nueva York" instead of "New York City." This is a future research direction.

8. Conclusion and Future Work

We have explained the basic concepts and principles of abstract syntax and its use as an interlingual representation as implemented in the Grammatical Framework (GF). We have also outlined a number of projects where GF is used in NLP tasks, emphasizing recent developments toward wide coverage. To conclude the discussion, we list some of the projects again, grouped according to their maturity:

Established, stable, widely used, several publications

- the GF programming language
- grammar compiler and runtime systems
- the resource grammar library (RGL)
- CNL translation and analysis
- natural language generation (NLG)
- logical semantics
- question answering
- commercial applications of CNL, NLG, and semantics

The core components of GF—the programming language and the RGL—have their limitations, as discussed in Sections 3.6 and 4.6. At the same time, they are well established and solid tools used in numerous projects and also by several companies.[42] The core GF has a clear identity, and radical changes in it would make it to something else than GF. It tries to follow the principle "do one thing and do it well"—the one thing being a tool for multilingual grammars with common abstract syntaxes. One still unexploited application of the core functionalities is the type-theoretical modeling of anaphora and other text-size phenomena (cf. Section 3.5).

Emerging, published, active

- multilingual WordNet-based concept extraction
- FrameNet-based abstraction to GF RGL
- multilingual verbalization of Abstract Meaning Representations (AMR)
- data augmentation and bootstrapping
- ud2gf pipelines
- Explainable Machine Translation (XMT)

42 We have found at least 15 companies, some of which are listed on `http://digitalgrammars.com`.

WordNet, FrameNet, and AMR are approaches to crosslingual meaning representations, and hence share some of the goals of GF. The resources created in them can be converted to GF grammar modules. In return, GF can complete them by providing precise and formalized syntax and morphology. GF can also help in augmenting and bootstrapping resources. The clearest example is treebanks, which can be generated either from grammars alone or by varying previously given trees. To find good combinations of hand-annotated and generated data is one of the most active directions for future work.

A related future task is semi-automatic creation of a wide-coverage multilingual GF lexicon of named entities. Such extension to RGL would be beneficial for various practical applications, including media monitoring use cases, for instance. Although it is rather trivial to generate a GF named entity lexicon for English (e.g., from the Wikipedia/DBpedia and Wikidata data sets), as named entities are not inflected in English, it is challenging in the case of inflectional languages. Another open question is the scope and coverage of such a lexicon: common and well-established entities vs. emerging entities of global or regional importance. The RGL could provide a comprehensive core lexicon of common entities, which could be extended by domain-specific applications.

Tested, published, but not yet widely used

- hybrid GF-SMT systems
- Construction Grammar implementations
- typological studies

Hybrid GF-SMT systems had a promising start (Enache et al. 2012), but were not continued when SMT itself was replaced by NMT in mainstream MT. SMT phrase alignment techniques are still a useful tool for bootstrapping translation lexica for GF. The main direction in current work, however, is to relate GF with neural techniques. Using neural UD parsing as a robust front end is a first step in this direction, and also a way to make MT more explainable (Section 2). An interesting next step would be to train neural parsers that directly return GF trees, without going via UD trees.

Grammatical constructions are needed to guarantee meaning-preserving compositional translation: It cannot scale up by word senses and syntactic structures alone. The challenge is how to identify these constructions. Phrase alignment can help in this, but a more general approach would extract abstract syntax functions and their linearizations from parallel texts. Initial experiments for such **concept alignment** align subtrees of abstract syntax trees obtained by wide coverage parsing, or—more robustly—parallel UD trees, which are then converted to GF. Preliminary results have been obtained from parallel GDPR data (Section 5.4; Ranta 2018), but they still need a lot of manual post-processing.

The potential for typological studies comes from the very idea of GF, which is to maintain an explicit distinction between shared and differing parts of grammars. One idea would be to elaborate the "questionnaire" technique in the LinGO Matrix: to implement a new language proceeds by answering a set of questions (Bender et al. 2010). The questions are posed by the abstract syntax, and the answers end up in the concrete syntax.

The Community. For any approach based on linguistic knowledge, an open culture for creating and sharing knowledge is essential. An important factor in building and

expanding the RGL has been the series of summer schools training new contributors. Except for the commercial applications, all of the projects mentioned in this article are open-source and available via public repositories. The grammar compiler is distributed under GPL (GNU General Public Licence), and the libraries and run-time under LGPL (Lesser General Public Licence) and BSD (Berkeley Software Distribution).

Acknowledgments

The authors want to thank the anonymous referees for their substantial and insightful comments. This work has received funding from Swedish Research Council under the grants Reliable Multilingual Digital Communication (2012-5746) and General Framework for Multilingual Text Processing (2012-4506) as well as from European Regional Development Fund under the grant agreement No. 1.1.1.1/16/A/219.

References

Agirre, Eneko, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. 2009. A study on similarity and relatedness using distributional and Wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, NAACL '09*, pages 19–27, Stroudsburg, PA.

Aho, A., M. Lam, R. Sethi, and J. Ullman. 2006. *Compilers: Principles, Techniques, and Tools. Second Edition*, Addison-Wesley.

Alshawi, H. 1992. *The Core Language Engine*. MIT Press, Cambridge, MA.

Angelov, K. 2011. *The Mechanics of the Grammatical Framework*, Ph.D. thesis, Chalmers University of Technology.

Angelov, K., B. Bringert, and A. Ranta. 2009. PGF: A portable run-time format for type-theoretical grammars. *Journal of Logic, Language and Information*, 19:201–228.

Angelov, K., B. Bringert, and A. Ranta. 2014. 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*, pages 41–44, Gothenburg.

Angelov, K. and P. Ljunglöf. 2014. Fast statistical parsing with parallel multiple context-free grammars. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 368–376, Gothenburg.

Angelov, Krasimir. 2009. Incremental parsing with parallel multiple context-free grammars. In *EACL'09*, pages 69–76, Athens.

Angelov, Krasimir. 2014. Bootstrapping open-source English-Bulgarian Computational dictionary. In *LREC*, pages 1018–1023.

Angelov, Krasimir. 2015. Orthography engineering in Grammatical Framework. In *Proceedings of the Grammar Engineering Across Frameworks (GEAF) 2015 Workshop*, pages 33–40, Beijing.

Angelov, Krasimir and Gleb Lobanov. 2016. Predicting translation equivalents in linked Wordnets. In the *26th International Conference on Computational Linguistics (COLING 2016)*, pages 26–32, Osaka.

Appel, A. 1998. *Modern Compiler Implementation in ML*. Cambridge University Press.

Bäckström, Linnéa, Benjamin Lyngfelt, and Emma Sköldberg. 2014. Towards interlingual constructicography. On correspondence between constructicon resources for English and Swedish. *Frames, Constructions and Computation. Special issue of Constructions and Frames*, 6(1):9–33.

Banarescu, Laura, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia.

Barzdins, Guntis. 2014. FrameNet CNL: A knowledge representation and information extraction language. In *Controlled Natural Language*, volume 8625 of *Lecture Notes in Computer Science*, pages 90–101. Springer.

Beesley, K. and L. Karttunen. 2003. *Finite State Morphology*. CSLI Publications.

Bender, E. M. and D. Flickinger. 2005. Rapid prototyping of scalable grammars: Towards modularity in extensions to a language-independent core. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing IJCNLP-05 (Posters/Demos)*, pages 203-208, Jeju Island.

Bender, Emily M., Scott Drellishak, Antske Fokkens, Michael Wayne Goodman, Daniel P. Mills, Laurie Poulson, and Safiyyah Saleem. 2010. Grammar prototyping and testing with the LinGO grammar matrix customization system. In *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, System Demonstrations*, pages 1–6, Uppsala.

Bernardy, Jean Philippe and Stergios Chatzikyriakidis. 2017. A type-theoretical system for the FraCaS test suite: Grammatical Framework meets Coq. In *IWCS 2017*, Montpellier, `www.aclweb.org/anthology/W17-6801/`.

Bloksma, Laura, Pedro Luis Diez-Orzas, and Vossen Piek. 1996. User requirements and functional specification of EuroWordNet project. Deliverable D001, WP1, EuroWordNet, LE2-4003.

Bond, Francis and Kyonghee Paik. 2012. A survey of Wordnets and their licenses. In *Proceedings of the 6th Global WordNet Conference*, (GWC 2012), pages 64–71, Matsue.

Borin, L., M. Forsberg and L. Lönngren. 2009. SALDO 1.0 (Svenskt associationslexikon version 2). Språkbanken, Göteborgs universitet. `http://spraakbanken.gu.se/personal/markus/publications/saldo_1.0.pdf`.

Borin, Lars, Dana Dannélls, and Normunds Grūzītis. 2018. Linguistics vs. language technology in constructicon building and use. In *Constructicography. Constructicon Development Across Languages*. John Benjamins, pages 229–254.

Borin, Lars, Markus Forsberg, and Lennart Lönngren. 2013. SALDO: A touch of yin to WordNet's yang. *Language Resources and Evaluation*, 47(4):1191–1211.

Bresnan, J., editor. 1982. *The Mental Representation of Grammatical Relations*, MIT Press.

Bringert, B. and A. Ranta. 2008. A pattern for almost compositional functions. *The Journal of Functional Programming*, 18(5–6):567–598.

Butt, M., H. Dyvik, T. Holloway King, H. Masuichi, and C. Rohrer. 2002. The parallel grammar project. In *COLING 2002, Workshop on Grammar Engineering and Evaluation*, pages 1–7, Taipei.

Camilleri, John J. 2013. A computational grammar and lexicon for Maltese. Master's thesis, Gothenburg, Sweden. September.

Chandioux, J. 1976. MÉTÉO: un système opérationnel pour la traduction automatique des bulletins météreologiques destinés au grand public. *META*, 21:127–133.

Chiarcos, Christian, Sebastian Hellmann, and Sebastian Nordhoff. 2012. Linking linguistic resources: Examples from the Open Linguistics Working Group. In Christian Chiarcos, Sebastian Nordhoff, and Sebastian Hellmann, editors, *Linked Data in Linguistics: Representing and Connecting Language Data and Language Metadata*. Springer, Berlin, Heidelberg, pages 201–216.

Chomsky, N. 1957. *Syntactic Structures*. Mouton, The Hague.

Dada, A. El and A. Ranta. 2006. Implementing an open source Arabic resource grammar in GF. In *20th Arabic Linguistics Symposium*. Kalamazoo, MI, pages 209–231.

Damonte, Marco and B. Cohen Shay. 2018. Cross-lingual Abstract Meaning Representation parsing. In *Proceedings of NAACL-HLT 2018*, New Orleans, LA.

Dannélls, Dana and Normunds Gruzitis. 2014. Extracting a bilingual semantic grammar from FrameNet-annotated corpora. In *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC)*, pages 2466–2473, Reykjavik.

Détrez, G. and A. Ranta. 2012. Smart paradigms and the predictability and complexity of inflectional morphology. In *EACL 2012*. pp. 645–653.

Diderichsen, P. 1962. *Elementær dansk grammatik*. Gyldendal, København.

Dorr, Bonnie J. 1994. Machine translation divergences: A formal description and proposed solution. *Computational Linguistics*, 20(4):597–633.

Dymetman, M. V. Lux and A. Ranta. 2000. XML and multilingual document authoring: Convergent trends. In *Proceedings of Computational Linguistics COLING*, pages 243–249, Saarbrücken.

Enache, Ramona, Cristina España Bonet, Aarne Ranta, Lluís Màrquez Villodre, et al. 2012. A hybrid system for patent translation. In *Proceedings of the 16th Annual Conference of the European Association for Machine Translation: EAMT 2012*, pages 269–278, Trento, Italy, May 28th-30th.

Erdenebadrakh, Nyamsuren. 2015. *Implementierung der Grammatik des modernen Mongolischen in der funktionalen Programmiersprache "Grammatical Framework,"* Ph.D. thesis CIS, LMU Munich.

Fellbaum, Christiane. 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

Fillmore, Charles J. 1985. Frames and the semantics of understanding. *Quaderni di Semantica*, 6(2):222–254.

Fillmore, Charles J. and Collin F. Baker. 2009. A frames approach to semantic analysis, In *The Oxford Handbook of Linguistic Analysis*, Oxford University Press, pages 313–340.

Fillmore, Charles J., Christopher R. Johnson, and Miriam R. L. Petruck. 2003. Background to FrameNet. *International Journal of Lexicography*, 16(3):235–250.

Flanigan, Jeffrey, Chris Dyer, Noah A. Smith, and Jaime Carbonell. 2016. Generation from Abstract Meaning Representation using tree transducers. In *Proceedings of NAACL-HLT 2016*, pages 731–739, San Diego, CA.

Flickinger, Dan. 2010. Accuracy vs. robustness in grammar engineering. In *Readings in Cognitive Science: Papers in Honor of Tom Wasow*, CSLI Publications, Stanford.

Forcada, M., M. Ginesti-Rosell, J. Nordfalk, J. O'Regan, S. Ortiz-Rojas, J. Perez-Ortiz, F. Sanchez-Martinez, G. Ramirez-Sanchez, and F. Tyers. 2011. Apertium: A free/open-source platform for rule-based machine translation. *Machine Translation*, 24:1–18.

Forsberg, M. and A. Ranta. 2004. Functional morphology. In *ICFP 2004*, pages 213–223, Showbird, UT.

Fuchs, Norbert E., Kaarel Kaljurand, and Tobias Kuhn. 2008, Attempto Controlled English for knowledge representation, *Reasoning Web, Fourth International Summer School 2008*, number 5224 in LNCS, pages 104–124, Springer.

Gazdar, G., E. Klein, G. Pullum, and I. Sag. 1985. *Generalized Phrase Structure Grammar*. Basil Blackwell, Oxford.

Gilardi, Luca and Collin Baker. 2018. Learning to align across languages: Toward multilingual FrameNet. *International FrameNet Workshop 2018: Multilingual FrameNets and Constructicons*, pages 13–22, Miyazaki.

Goldberg, Adele E. 2013, Constructionist approaches. In *The Oxford Handbook of Construction Grammar*. Oxford University Press.

Gruzitis, Normunds and Dana Dannélls. 2017. A multilingual FrameNet-based grammar and lexicon for Controlled Natural Language. *Language Resources and Evaluation*, 51(1):37–66.

Gruzitis, Normunds, Dana Dannélls, Benjamin Lyngfelt, and Aarne Ranta. 2015. Formalising the Swedish Constructicon in Grammatical Framework. In *Proceedings of the ACL Workshop on Grammar Engineering Across Frameworks (GEAF)*, pages 49–56, Beijing.

Gruzitis, Normunds, Didzis Gosko, and Guntis Barzdins. 2017. RIGOTRIO at SemEval-2017 Task 9: Combining machine learning and grammar engineering for AMR parsing and generation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval)*, pages 924–928, Vancouver.

Gunning, D. 2017. Explainable Artificial Intelligence (XAI). DARPA Program Information, `https://www.darpa.mil/program/explainable-artificial-intelligence`.

Harper, R., F. Honsell, and G. Plotkin. 1993. A framework for defining logics. *JACM*, 40(1):143–184.

Hublet, François. 2019. IDL-PMCFG, a grammar formalism for describing free order languages. Under submission.

Hutchins, W. J. and H. L. Somers. 1992. *An Introduction to Machine Translation*. Academic Press Limited, London.

Janda, Laura A., Olga Lyashevskaya, Tore Nesset, Ekaterina Rakhilina, and Francis M. Tyers. 2018. A constructicon for Russian. In *Constructicography: Constructicon Development Across Languages*. John Benjamins, pages 165–181.

Johannisson, K. 2005. *Formal and Informal Software Specifications*. Ph.D. thesis, Dept.of Computing Science, Chalmers University of Technology and Gothenburg University.

Joshi, A. 1985, Tree-adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Parsing*, Cambridge University Press, pages 206–250.

Joshi, Aravind K. and Yves Schabes. 1997. Tree-adjoining grammars. In *Handbook of Formal Languages*. Springer, 69–123.

Kaljurand, Kaarel and Tobias Kuhn. 2013. A multilingual semantic wiki based on Attempto controlled English and Grammatical Framework. In *The Semantic Web: Semantics and Big Data*, Springer. pages 427–441.

Kalldal, Oscar and Maximilian Ludvigsson. 2017. Unsupervised disambiguation of abstract syntax. Master's thesis, Chalmers University of Technology.

Kallmeyer, L. 2010. *Parsing Beyond Context-Free Grammars*. Springer Publishing Company, Incorporated.

Kallmeyer, Laura and Wolfgang Maier. 2013. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*, 39(1):87–119.

Kamholz, David, Jonathan Pool, and Susan Colowick. 2014. Panlex: Building a resource for panlingual lexical translation. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, European Language Resources Association (ELRA), Reykjavik, pp. 3145–3150.

Kann, Viggo and Joachim Hollman. 2011. Slutrapport för projektet Folkets engelsk-svenska lexikon. Royal Institute of Technology in Stockholm, `https://folkets-lexikon.csc.kth.se/folkets/projekt/slutrapport.pdf`.

Kaplan, R. and M. Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20:331–380.

Karttunen, L. and M. Kay. 1985. Parsing in a free word order language. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Parsing, Psychological, Computational, and Theoretical Perspectives*. Cambridge University Press, pages 279–306.

Kay, Martin. 2017. *Translation Linguistic and Philosophical Perspectives*, CSLI Publications, Stanford.

Khegai, J. 2006. GF parallel resource grammars and Russian. In *Coling/ACL 2006*, pages 475–482.

Kituku, B., W. Nganga, and L. Muchemi. 2019. Towards Kikamba computational grammar. *Journal of Data Analysis and Information Processing*, 7:250–275.

Koehn, P. 2010. *Statistical Machine Translation*, Cambridge University Press.

Koeva, G., Totkov S., and A. Genov. 2004. Towards Bulgarian Wordnet. *Romanian Journal of Information Science and Technology*, 7(1–2):45–61.

Kolachina, P. and A. Ranta. 2015. GF wide-coverage English-Finnish MT system for WMT 2015. *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 141–144, Lisbon.

Kolachina, P. and A. Ranta. 2016. From abstract syntax to universal dependencies. *Linguistic Issues in Language Technology*, 13:1–57.

Kolachina, Prasanth and Aarne Ranta. 2019. Bootstrapping UD treebanks for delexicalized parsing. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, pages 15–24.

Kuhn, T. 2014. A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170.

Lee, J., K. Cho, and T. Hofmann. 2016. Fully character-level neural machine translation without explicit segmentation. *CoRR*, abs/1610. 03017.

Lindén, Krister and Lauri Carlson. 2010. FinnWordNet - WordNet på finska via översättning. *LexicoNordica - Nordic Journal of Lexicography*, 17:119–140.

Listenmaa, Inari and Kaarel Kaljurand. 2014. 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, p 13.

Ljunglöf, P. 2004. *The Expressivity and Complexity of Grammatical Framework*. Ph.D. thesis, Department of Computing Science, Chalmers University of Technology and Gothenburg University.

Lu, Yichao, Phillip Keung, Faisal Ladhak, Vikas Bhardwaj, Shaonan Zhang, and Jason Sun. 2018. A neural interlingua for multilingual machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 84–92. Brussels.

Lyngfelt, Benjamin. 2018. Introduction: Constructicons and constructicography. In *Constructicography: Constructicon Development Across Languages*. John Benjamins, pages 1–18.

Lyngfelt, Benjamin, Tiago T. Torrent, Adrieli Laviola, Linnéa Bäckström, Anna H. Hannesdóttir, and Ely Edison da Silva Matos. 2018. Aligning constructicons across languages: A trilingual comparison between English, Swedish, and Brazilian Portuguese. In *Constructicography: Constructicon Development Across Languages*. John Benjamins, pages 255–302.

Marcus, M. P., B. Santorini and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Martin-Löf, P. 1984. *Intuitionistic Type Theory*, Bibliopolis, Napoli.

May, Jonathan. 2016. Semeval-2016 Task 8: Meaning representation parsing. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval)*, pages 1063–1073, San Diego, CA.

May, Jonathan and Jay Priyadarshi. 2017. Semeval-2017 Task 9: Abstract meaning

representation parsing and generation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval)*, pages 536–545, Vancouver.

McConnell, R. M., K. Mehlhorn, S. Näher, and P. Schweitzer. 2011. Certifying algorithms. *Computer Science Review*, 5(2):119–161.

Mel'cuk, I. 1997. Vers une linguistique Sens-Texte. Leçon inaugurale. `http://olst.ling.umontreal.ca/pdf/MelcukColldeFr.pdf`.

Montague, R. 1974. *Formal Philosophy*, Yale University Press, New Haven. Collected papers edited by Richmond Thomason.

Mäenpää, P. and A. Ranta. 1999. The type theory and type checker of GF. In *PLI-1999: Workshop on Logical Frameworks and Meta-languages*, Paris.

Ng'ang'a, Wanjiku. 2012. Building Swahili resource grammars for the Grammatical Framework. In Diana Santos, Krister Lindén, and Wanjiku Ng'ang'a, editors, *Shall We Play the Festschrift Game? Essays on the Occasion of Lauri Carlson's 60th Birthday*. Springer, Berlin, Heidelberg, pages 215–226.

Nivre, Joakim, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sa mpo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 1659–1666, European Language Resources Association (ELRA), Portorož.

Oepen, S., D. Flickinger K. Toutanova, and C. D. Manning. 2004. LinGO Redwoods, a rich and dynamic treebank for HPSG. *Research on Language and Computation*, 2:575–596.

Papadopoulou, Ioanna. 2013. GF modern Greek resource grammar. In *Proceedings of the Student Research Workshop associated with RANLP 2013*, pages 126–133, Hissar.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. *ACL*, pages 311–318.

Pollard, C. and I. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Pretorius, Laurette, Laurette Marais, and Ansu Berg. 2017. A GF miniature resource grammar for Tswana: Modelling the proper verb. *Language Resources and Evaluation*, 51(1):159–189.

Ranta, A. 1994. *Type Theoretical Grammar*. Oxford University Press.

Ranta, A. 2004a. Computational semantics in type theory. *Mathematics and Social Sciences*, 165:31–57.

Ranta, A. 2004b. Grammatical framework: A type-theoretical grammar formalism. *The Journal of Functional Programming*, 14(2):145–189.

Ranta, A. 2007. Modular grammar engineering in GF. *Research on Language and Computation*, 5:133–158.

Ranta, A. 2009a. *Grammars as software libraries*. In *From Semantics to Computer Science. Essays in Honour of Gilles Kahn*, pages 281–308. Cambridge University Press.

Ranta, A. 2009b. The GF resource grammar library. *Linguistics in Language Technology*, 2, pages 1–65.

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

Ranta, A. 2012. On the syntax and translation of Finnish discourse clitics. In *Shall We Play the Festschrift Game? Essays on the Occasion of Lauri Carlson's 60th Birthday*. Springer Berlin Heidelberg, pages 227–241.

Ranta, A. and K. Angelov. 2010. Implementing controlled languages in GF. In *Proceedings of CNL-2009, Marettimo Island*, volume 5972 of LNCS, pages 82–101.

Ranta, A., G. Détrez and R. Enache. 2012. Controlled language for everyday use: The MOLTO phrasebook. In *CNL 2012: Controlled Natural Language*, volume 7175 of LNCS/LNAI.

Ranta, A. and P. Kolachina. 2017. From universal dependencies to abstract syntax. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies*, pages 107–116, Gothenburg.

Ranta, Aarne. 2011b. Translating between language and logic: What is easy and what is difficult. *Automated Deduction – CADE-23*, pages 5–25, Springer Berlin Heidelberg, Berlin, Heidelberg.

Ranta, Aarne. 2014. Embedded controlled languages. In *Controlled Natural Language*, pages 1–7, Springer International Publishing.

Ranta, Aarne. 2017. Explainable machine translation with interlingual trees as certificates. In *Proceedings of the Conference on Logic and Machine Learning in Natural Language (LaML 2017)*, pages 63–78, Gothenburg.

Ranta, Aarne. 2018. Concept alignment for compositional translation (abstract). In

Proceedings of the Symposium on Logic and Algorithms in Computational Linguistics 2018 (LACompLing2018), page 17, Stockholm University DiVA Portal for digital publications.

Ranta, Aarne. 2019. What are grammars good for? Reflections on twenty years of Grammatical Framework. In Cleo Condoravdi and Tracy Holloway King, editors, *Tokens of Meaning: Papers in Honor of Lauri Karttunen*. CSLI Publications, Stanford, pages 543–566.

Ranta, Aarne, Qiao Haiyan, and Yan Tian. 2015. Chinese in the Grammatical Framework: Grammar, translation, and other applications. In *Proceedings of the Eighth SIGHAN Workshop on Chinese Language Processing, ACL*, pages 100–109, Beijing.

Ranta, Aarne, Yan Tian, and Haiyan Qiao. 2015. Chinese in the Grammatical Framework: Grammar, translation, and other applications. In *Proceedings of the Eighth SIGHAN Workshop on Chinese Language Processing, ACL*, pages 100–109, Beijing.

Ranta, Aarne, Christina Unger, and Daniel Vidal Hussey. 2015. Grammar engineering for a customer: A case study with five languages, pages 1–8, Beijing.

Rautio, Jussi and Maarit Koponen. 2013. D9.2 MOLTO evaluation and assessment report. The MOLTO Consortium, http://www.molto-project.eu/biblio/deliverable/d92-molto-evaluaton-and-assessment-report.

Rayner, M., D. Carter, P. Bouillon, V. Digalakis, and M. Wirén. 2000. *The Spoken Language Translator*. Cambridge University Press.

Riezler, Stefan, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell III, and Mark Johnson. 2002. Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 271–278.

Rosetta, M. T. 1994. *Compositional Translation*. Kluwer, Dordrecht.

Roth, Michael and Anette Frank. 2009. A NLG-based application for walking directions. In *Proceedings of the ACL-IJCNLP 2009 Software Demonstrations*, pages 37–40, Suntec.

Sahin, Gozde Gul and Mark Steedman. 2018. Data augmentation via dependency tree morphing for low-resource languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5004–5009. Brussels, Belgium.

Schuler, Karin Kipper. 2005. VerbNet: A broad-coverage, comprehensive verb lexicon. Ph.D thesis, Department of Computer Science, University of Pennsylvania.

Seki, H., T. Matsumura, M. Fujii, and T. Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

Sharoff, Serge and Joakim Nivre. 2011. The proper place of men and machines in language technology: Processing Russian without any linguistic knowledge. In *Computational Linguistics and Intellectual Technologies*, Moscow.

Shieber, Stuart M. and Yves Schabes. 1990. Synchronous tree-adjoining grammars. In *COLING*, pages 253–258.

Simov, Kiril and Petya Osenova. 2010. Constructing of an ontology-based lexicon for Bulgarian. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, European Language Resources Association (ELRA), Valletta. pp. 3840–3844.

Spevak, Olga. 2010. *Constituent Order in Classical Latin Prose*. John Benjamins Publishing, Amsterdam.

Stallman, Richard. 2001. *Using and Porting the GNU Compiler Collection*, Free Software Foundation.

Stamou, Sofia Oflazer, Kamel Pala, K. Christoudoulakis, et al. 2004. BalkaNet: A multilingual semantic network for Balkan languages. *Romanian Journal of Information Science and Technology*, 7(1–2):12–14.

Steedman, M. 1988. Combinators and grammars. In R. Oehrle, E. Bach, and D. Wheeler, editors, *Categorial Grammars and Natural Language Structures*. D. Ridel, Dordrecht, pages 417–442.

Steedman, M. 2000. *The Syntactic Process*. The MIT Press.

Straka, Milan and Jana Straková. 2017. Tokenizing, POS tagging, lemmatizing and parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver.

Swadesh, Morris. 1955. Towards greater accuracy in lexicostatistic dating. *International Journal of American Linguistics*, 21:121–137.

Tiedemann, Jörg and Željko Agić. 2016. Synthetic treebanking for cross-lingual

dependency parsing. *Journal of Artificial Intelligence Research (JAIR)*, 55:209–248.
Traustason, Bjarki. 2016. Open source Icelandic resource grammar in GF. Master's thesis, Chalmers University of Technology, Gothenburg, Sweden.
Vauquois, B. 1968. A survey of formal grammars and algorithms for recognition and transformation in mechanical translation. In *IFIP Congress (2)*, pages 1114–1122.
Viberg, øAke, Kerstin Lindmark, Ann Lindvall, and Ingmarie Mellenius. 2002. The Swedish WordNet project. *In Proceedings of Euralex*, pages 407–412. Copenhagen, Denmark.
Vijay-Shanker, K., David J. Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Stanford, CA.
Virk, Shafqat Mumtaz, K. V. S. Prasad, Aarne Ranta, and Krasimir Angelov. 2014. Developing an interlingual translation lexicon using WordNets and Grammatical Framework. *COLING 2014*, page 55.
Xue, Nianwen, Ondrej Bojar, Jan Hajic, Martha Palmer, Zdenka Uresova, and Xiuhong Zhang. 2014. Not an interlingua, but close: Comparison of English AMRs to Chinese and Czech. In *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC)*, Reykjavik. Vol. 14, pp. 1765–1772.
Zimina, Elizaveta. 2012. Fitting a round peg in a square hole: Japanese resource grammar in GF. *JapTAL, LNCS/LNAI*, Kanazawa.
Zimina, Elizaveta, Jyrki Nummenmaa, Kalervo Järvelin, Jaakko Peltonen, and Kostas Stefanidis. 2018. MuG-QA: Multilingual grammatical question answering for RDF data. In *SemWebEval@ESWC*.