Speech Translation with Grammatical Framework

Björn Bringert
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
bringert@chalmers.se

Abstract

Grammatical Framework (GF) is a grammar formalism which supports interlingua-based translation, library-based grammar engineering, and compilation to speech recognition grammars. We show how these features can be used in the construction of portable high-precision domain-specific speech translators.

1 Introduction

Speech translators for safety-critical applications such as medicine need to offer high-precision translation. One way to achieve high precision is to limit the coverage of the translator to a specific domain. The development of such high-precision domain-specific translators can be resource intensive, and require rare combinations of developer skills. For example, consider developing a Russian–Swahili speech translator for the orthopedic domain using direct translation between the two languages. Developing such a system could require an orthopedist programmer and linguist who speaks Russian and Swahili. Such people may be hard to find. Furthermore, developing translators for all pairs of N languages requires $O(N^2)$ systems, developed by an equal number of bilingual domain experts.

The language pair explosion and the need for the same person to possess knowledge about the source and target languages can be avoided by using an interlingua-based approach. The requirement that developers be both domain experts and linguists can be addressed by the use of grammar libraries which implement the domain-independent linguistic details of each language.

Grammatical Framework (GF) (Ranta, 2004) is a type-theoretic grammar formalism which is well suited to high-precision domain-specific interlingua-based translation (Khegai, 2006), and library-based grammar engineering (Ranta, 2008). GF divides grammars into *abstract syntax* and *concrete syntax*. The abstract syntax defines *what* can be said in the grammar, and the concrete syntax defines *how* it is said in a particular language. If one abstract syntax syntax is given multiple concrete syntaxes, the abstract syntax can be used as an interlingua. Given an abstract and a concrete syntax, GF allows both parsing (text to abstract syntax) and linearization (abstract syntax to text). This means that interlingua-based translation is just a matter of parsing in one language and linearizing to another.

The GF resource grammar library (Ranta, 2008) implements the domain-independent morphological and syntactic details of eleven languages. A grammar writer can use functions from a resource grammar when defining the concrete syntax of an application grammar. This is made possible by GF's support for *grammar composition*, and frees the grammar writer from having to implement linguistic details such as agreement, word order etc.

In addition to parsing and linearization, the declarative nature of GF grammars allows them to be compiled to other grammar formats. The GF speech recognition grammar compiler (Bringert, 2007) can produce context-free grammars or finite-state models which can be used to guide speech recognizers.

These components, interlingua-based translation, grammar libraries, and speech recognition grammar compilation, can be used to develop

Coling 2008: Proceedings of the workshop on Speech Processing for Safety Critical Translation and Pervasive Applications, pages 5–8
Manchester, August 2008

domain-specific speech translators based on GF grammars. Figure 1 shows an overview of a minimal unidirectional speech translator which uses these components. This is a proof-of-concept system that demonstrates how GF components can be used for speech translation, and as such it can hardly be compared to a more complete and mature system such as MedSLT (Bouillon et al., 2005). However, the system has some promising features compared to systems based on unification grammars: the expressive power of GF's concrete syntax allows us to use an application-specific interlingua without any transfer rules, and the wide language support of the GF Resource Grammar library makes it possible to quickly port applications to new languages.

In Section 2 we show a small example grammar for a medical speech translator. Section 3 briefly discusses how a speech translator can be implemented. Section 5 describes some possible extensions to the proof-of-concept system, and Section 6 offers some conclusions.

2 Example Grammar

We will show a fragment of a grammar for a medical speech translator. The example comes from Khegai's (2006) work on domain-specific translation with GF, and has been updated to use the current version of the GF resource library API.

The small abstract syntax (interlingua) shown in Figure 2 has three categories (**cat**): the start category Prop for complete utterances, Patient for identifying patients, and Medicine for identifying medicines. Each category contains a single function (**fun**). There are the nullary functions ShePatient and PainKiller, and the binary NeedMedicine, which takes a Patient and a Medicine as arguments, and produces a Prop. This simple abstract syntax only allows us to construct the term NeedMedicine ShePatient PainKiller. A larger version could for example include categories for body parts, symptoms and illnesses, and more functions in each category. An example of a term in such an extended grammar could be And (Injured TheyPatient Foot) (NeedMedicine HePatient Laxative).

For this abstract syntax we can use the English resource grammar to write an English concrete syntax, as shown in Figure 3. The resource grammar category NP is used as the linearization type (**lincat**) of the application grammar categories

```
abstract Health = {
flags startcat = Prop;
cat Patient; Medicine; Prop;
fun
  ShePatient : Patient;
  PainKiller  : Medicine;
  NeedMedicine : Patient → Medicine → Prop;
}
```

Figure 2: Example abstract syntax.

Patient and Medicine, and S is used for Prop. The linearizations (**lin**) of each abstract syntax function use overloaded functions from the resource grammar, such as $mkCl$ and mkN which create clauses and nouns, respectively.

```
concrete HealthEng of Health =
  open SyntaxEng, ParadigmsEng in {
lincat Patient, Medicine = NP; Prop = S;
lin
  ShePatient = mkNP she_Pron;
  PainKiller =
    mkNP indefSgDet (mkN "painkiller");
  NeedMedicine p m =
    mkS (mkCl p (mkV2 (mkV "need")) m);
}
```

Figure 3: English concrete syntax.

Figure 4 shows a Swedish concrete syntax created in the same way. Note that PainKiller in Swedish uses a mass noun construction rather than the indefinite article.

```
concrete HealthSwe of Health =
  open SyntaxSwe, ParadigmsSwe in {
lincat Patient, Medicine = NP; Prop = S;
lin
  ShePatient = mkNP she_Pron;
  PainKiller =
    mkNP massQuant
      (mkN "smärtstillande");
  NeedMed p m =
    mkS (mkCl p
      (mkV2 (mkV "behöver")) m);
}
```

Figure 4: Swedish concrete syntax.

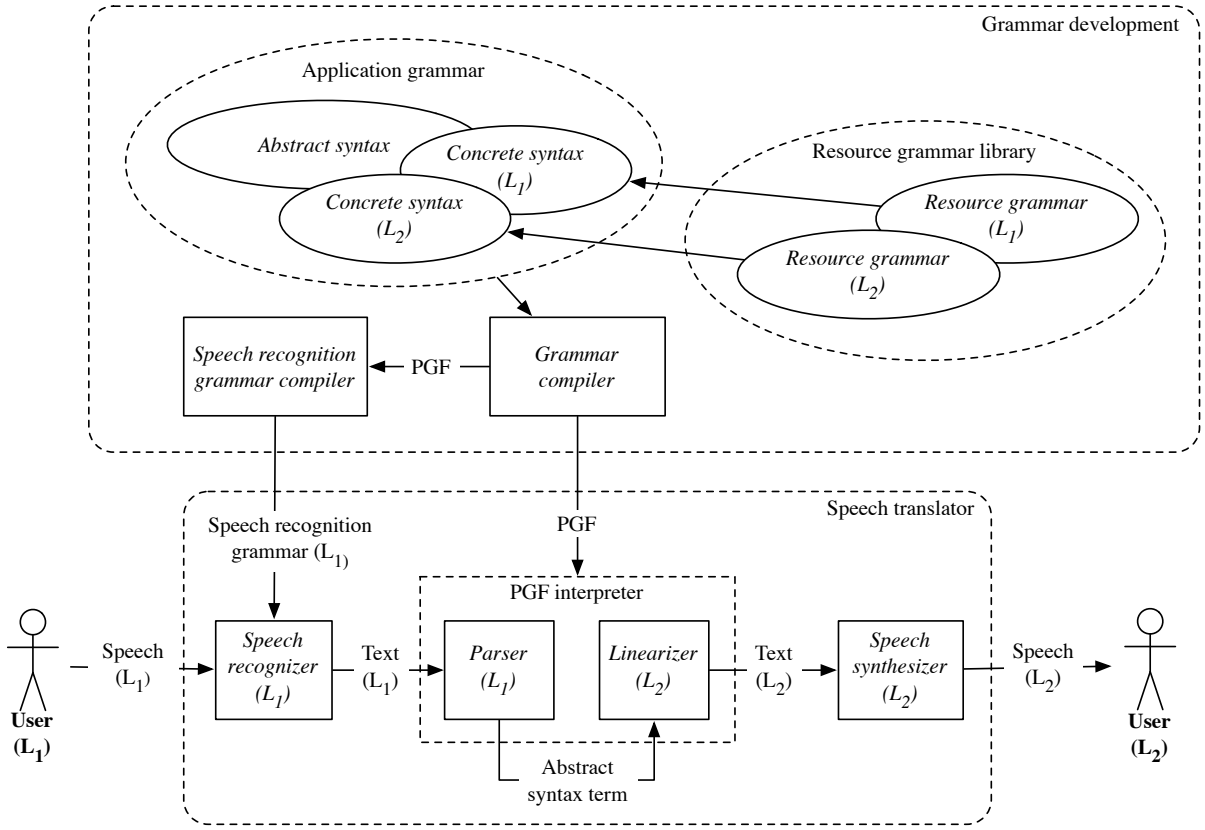

Figure 1: Overview of a GF-based speech translator. The developer writes a multilingual application grammar using the resource grammar library. This is compiled to a PGF (Portable Grammar Format) grammar used for parsing and linearization, and a speech recognition grammar. Off-the-shelf speech recognizers and speech synthesizers are used together with a PGF interpreter in the running system.

3 Speech Translator Implementation

The GF grammar compiler takes grammars in the GF source language used by programmers, and produces grammars in a low-level language (Portable Grammar Format, PGF (Angelov et al., 2008)) for which interpreters can be easily and efficiently implemented. There are currently PGF implementations in Haskell, Java and JavaScript. The GF speech recognition grammar compiler (Bringert, 2007) targets many different formats, including Nuance GSL, SRGS, JSGF and HTK SLF. This means that speech translators based on GF can easily be implemented on almost any platform for which there is a speech recognizer and speech synthesizer. We have run Java-based versions under Windows using Nuance Recognizer and RealSpeak or FreeTTS, Haskell-based versions under Linux using Nuance Recognizer and RealSpeak, and JavaScript-based prototypes in the Opera XHTML+Voice-enabled web browser on Zaurus PDAs and Windows desktops.

The speech translation system itself is domain-independent. All that is required to use it in a new domain is an application grammar for that domain.

4 Evaluation

Since we have presented a proof-of-concept system that demonstrates the use of GF for speech translation, rather than a complete system for any particular domain, quantitative translation performance evaluation would be out of place. Rather, we have evaluated the portability and speed of prototyping. Our basic speech translators written in Java and Haskell, using existing speech components and PGF interpreters, require less than 100 lines of code each. Developing a small domain for the translator can be done in under 10 minutes.

5 Extensions

5.1 Interactive Disambiguation

The concrete syntax for the source language may be ambiguous, i.e. there may be sentences for which parsing produces multiple abstract syntax

terms. The ambiguity can sometimes be preserved in the target language, if all the abstract syntax terms linearize to the same sentence.

In cases where the ambiguity cannot be preserved, or if we want to force disambiguation for safety reasons, we can use a *disambiguation grammar* to allow the user to choose an interpretation. This is a second concrete syntax which is completely unambiguous. When the user inputs an ambiguous sentence, the system linearizes each of the abstract syntax terms with the disambiguation grammar, and prompts the user to select the sentence with the intended meaning. If only some of the ambiguity can be preserved, the number of choices can be reduced by grouping the abstract syntax terms into equivalence classes based on whether they produce the same sentences in the target language. Since all terms in a class produce the same output, the user only needs to select the correct class of unambiguous sentences.

Another source of ambiguity is that two abstract syntax terms can have distinct linearizations in the source language, but identical target language linearizations. In this case, the output sentence will be ambiguous, even though the input was unambiguous. This could be addressed by using unambiguous linearizations for system output, though this may lead to the use of unnatural constructions.

5.2 Bidirectional Translation

Since GF uses the same grammar for parsing and linearization, the grammar for a translator from L_1 to L_2 can also be used in a translator from L_2 to L_1, provided that the appropriate speech components are available. Two unidirectional translators can be used as a bidirectional translator, something which is straightforwardly achieved using two computers. While PGF interpreters can already be used for bidirectional translation, a single-device bidirectional speech translator requires multiplexing or duplicating the sound hardware.

5.3 Larger Input Coverage

GF's *variants* feature allows an abstract syntax function to have multiple representations in a given concrete syntax. This permits some variation in the input, while producing the same interlingua term. For example, the linearization of PainKiller in the English concrete syntax in Figure 3 could be changed to:

$mkNP\ indefSgDet\ (\mathbf{variants}\{$
  mkN "painkiller"; mkN "analgesic"$\});$

6 Conclusions

Because it uses a domain-specific interlingua, a GF-based speech translator can achieve high precision translation and scale to support a large number of languages.

The GF resource grammar library reduces the development effort needed to implement a speech translator for a new domain, and the need for the developer to have detailed linguistic knowledge.

Systems created with GF are highly portable to new platforms, because of the wide speech recognition grammar format support, and the availability of PGF interpreters for many platforms.

With additional work, GF could be used to implement a full-scale speech translator. The existing GF components for grammar development, speech recognition grammar compilation, parsing, and linearization could also be used as parts of larger systems.

References

Angelov, Krasimir, Björn Bringert, and Aarne Ranta. 2008. PGF: A Portable Run-Time Format for Type-Theoretical Grammars. Manuscript, `http://www.cs.chalmers.se/~bringert/publ/pgf/pgf.pdf`.

Bouillon, P., M. Rayner, N. Chatzichrisafis, B. A. Hockey, M. Santaholma, M. Starlander, H. Isahara, K. Kanzaki, and Y. Nakao. 2005. A generic Multi-Lingual Open Source Platform for Limited-Domain Medical Speech Translation. pages 5–58, May.

Bringert, Björn. 2007. Speech Recognition Grammar Compilation in Grammatical Framework. In *Proceedings of the Workshop on Grammar-Based Approaches to Spoken Language Processing*, pages 1–8, Prague, Czech Republic.

Khegai, Janna. 2006. Grammatical Framework (GF) for MT in sublanguage domains. In *Proceedings of EAMT-2006, 11th Annual conference of the European Association for Machine Translation, Oslo, Norway*, pages 95–104, June.

Ranta, Aarne. 2004. Grammatical Framework: A Type-Theoretical Grammar Formalism. *Journal of Functional Programming*, 14(2):145–189, March.

Ranta, Aarne. 2008. Grammars as software libraries. In Bertot, Yves, Gérard Huet, Jean-Jacques Lévy, and Gordon Plotkin, editors, *From semantics to computer science: essays in honor of Gilles Kahn*. Cambridge University Press.