Implementing Controlled Languages in GF

Aarne Ranta and Krasimir Angelov

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

Abstract. The paper introduces GF, Grammatical Framework, as a tool for implementing controlled languages. GF provides a high-level grammar formalism and a resource grammar library that make it easy to write grammars that cover similar fragments in several natural languages at the same time. Authoring help tools and automatic translation are provided for all grammars. As an example, a grammar of Attempto Controlled English is implemented and then ported to French, German, and Swedish.

1 Introduction

Controlled languages are fragments of natural languages. They are designed to be clear and unambiguous. If they are mechanically processable, they have the benefits of formal languages; moreover, since they look like natural languages, humans can understand them without training. However, *writing* controlled language can be a problem, since the author has to remain within a fragment whose limits she has to learn. Moreover, *implementing* a controlled language on a computer requires a considerable effort, since the syntax of even small fragments of natural languages is much more complex than the syntax of formal languages.

This paper addresses both the writing and the implementation problem, by using GF (Grammatical Framework, Ranta 2004). GF is a multilingual grammar formalism, which uses type-theoretical representations of semantic content and reversible mappings from the content to various languages. These mappings work for both parsing and generation; composing them gives automatic translation. The parsers moreover support *completion*, which displays grammatically correct continuations to the user, thereby helping her to stay within the fragment.

Writing grammars in GF is helped by the *resource grammar library* (Ranta 2007), a set of wide-coverage grammars currently implemented for 12 languages: Bulgarian, Catalan, Danish, English, Finnish, French, German, Italian, Norwegian, Russian, Spanish, and Swedish. The library takes care of linguistic details (inflection, agreement, word order), allowing the grammarian to concentrate on the content.

Starting with experiments at Xerox (Dymetman & al. 2000), GF has been used in numerous natural language applications: mathematical proof editing (Hallgren and Ranta 2000), software specification authoring (Hähnle & al. 2005), spoken dialogue systems (Ranta and Cooper 2004, Lemon and Liu 2006, Larsson and Ljunglöf 2008), translation of mathematical exercises (Caprotti 2006), multilingual Wiki pages (Meza Moreno and Bringert 2008), and multilingual generation in the cultural heritage domain

(Dannélls 2008). Many of these applications can be described as domain-specific controlled languages, but some of them are not controlled in all senses of the word; they may, for instance, permit ambiguous input.

In this paper, we will present a GF grammar for Attempto Controlled English (ACE, Attempto 2008). We first implemented English as specified in the ACE documentation, and then ported it to French German, and Swedish. Thus we can now translate Attempto documents to all these languages, and use these languages to give input to Attempto applications. We can also combine GF authoring tools with Attempto tools.

2 Attempto in GF

Our implementation of Attempto in GF follows the document "ACE 6.0 Construction Rules" (Attempto 2008). The current grammar includes 98% of the syntactic constructions reported there, as well as a lexicon containing the words that are used in the examples in this document. The content word lexicon is a separate module that can easily be replaced by the large Attempto lexicon, or some domain-specific terminology lexicon. The more formal document "ACE 6.0 Syntax Report" (Attempto 2008) was used as a reference, but we did not follow it in detail; the main reason was that the Syntax Report follows English too closely to be generalizable in the way we wanted.

Following standard computer science concepts, GF distinguishes between *abstract syntax* and *concrete syntax*. The abstract syntax is a system of *trees*, which encode the structures of the language. It is up to the grammarian to decide if the tree structure is semantic or syntactic, or something in between. As we followed the Attempto construction rules, we ended up with structures with a rather syntactic flavour. In the resulting grammar, the English sentence *every rich customer is important* receives the tree

```
(vpS (everyNP (adjCN (positAP rich_A) customer_CN))
     (apVP (positAP important_A)))
```

Trees in GF are like Lisp terms: a tree is a *function* applied to zero or more trees; unlike Lisp, GF trees are statically typed. The outermost function in the example tree is

```
vpS : NP -> VP -> S
```

which takes a noun phrase (`NP`) and a verb phrase (`VP`) to a sentence (`S`). An abstract syntax in GF is a set of such functions and *categories* (basic types, such as `NP`); our Attempto implementation has 28 categories and 120 (non-lexical) functions.

A concrete syntax defines the *linearizations* of trees into *strings* and, more generally, *feature structures*. Features are used for phenomena such as agreement and word order. What features are needed depends heavily on language; the crucial property of GF (as opposed to most grammar formalisms) is that the features are treated separately in each concrete syntax, which enables a language-independent abstract syntax.

Getting all details right in concrete syntax is highly non-trivial. This is where the GF Resource Grammar Library comes in. When using the library, concrete syntax is not written by using strings and features explicitly, but by calling functions in the library. Thus the linearization of the predication rule of Attempto is simply

```
  lin vpS np vp = mkS presTense posPol (mkCl np vp)
```

The function `mkCl` takes a noun phrase and a verb phrase and builds a clause, whose tense and polarity are not yet fixed. The function `mkS` is used to make it a present-tense sentence with positive polarity. The grammar also uses `mkS` with the argument `negPol` to express Attempto's negative predication.

The abstract syntax rules in the library are exactly the same for all 12 languages. The Attempto syntax can therefore be implemented by using a shared *parametrized module* (Ranta 2007). For example the German version is implemented as:

```
concrete AttemptoGer of Attempto = SymbolsC ** AttemptoI with
  (Syntax = SyntaxGer),
  (Symbolic = SymbolicGer),
  (LexAttempto = LexAttemptoGer) ;
```

Here the main implementation module is `AttemptoI`. It provides the main functionality but leaves some grammatical details unspecified. In fact in the implementation it uses interfaces like `Syntax` and `Symbolic` as black boxes which are supposed to deal with low-level details like the word order or the agreement in the target language. When concrete language is selected (German in this case) the real implementations are specified with the `with` clause. In this case the implementations for `SyntaxGer` and `SymbolicGer` come from the resource library.

This is not the case for the lexicon. Each language in the library has its own module of inflection paradigms, since different pieces of information are needed to define the inflection tables of words. To give an example, our Attempto lexicon has the abstract syntax constant `customer_N`, which is linearized as follows:

```
  mkN "customer"                  -- English
  mkN "client"                    -- French
  mkN "Kunde" "Kunden" masculine  -- German
  mkN "kund" "kunder"             -- Swedish
```

Now we can map the example tree to English, French, German, and Swedish:

every rich customer is important
chaque client riche est important
jeder reiche Kunde ist wichtig
varje rik kund är viktig

If syntactic combinations are implemented with a parametrized module, as in the case of our Attempto grammar, adding a new language to a multilingual GF system typically requires only a new implementation of the lexicon.

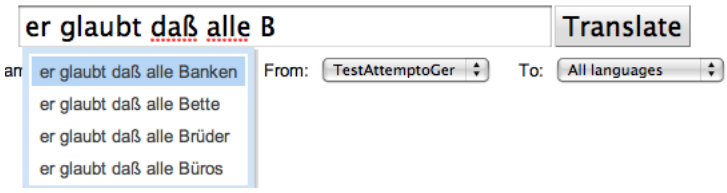
In some cases, however, some changes are needed. It may happen that the new language uses two different words in different contexts to express something that was a single word before, just because it happened to be linearized in the same way in the language that was the model of the first abstract syntax. This problem can mostly be

avoided when the development is guided by some language independent ontology, instead of a particular language; this approach seems to be used in Attempto applications as well.

Another case in which more work is needed than just a new lexicon is when the new target language uses different syntactic structures to express a certain combination than those defined in the parametrized module. This problem is unavoidable, but can be solved by simply overriding the parametrized solution in the language in question. In our Attempto grammars, we didn't use this possibility, but retained the same constructions in all languages.

3 Authoring Tools

Most controlled language systems provide some kinds of authoring tools which guides the user how to stay within the scope of the grammar. The following picture shows a translator client using a GF web server. It has a completion parser, which here shows grammatically correct completions of the letter *B* in the middle of a German sentence. The system uses the parsing algorithm described in (Angelov 2009) and the web application toolkit described in (Bringert & al. 2009).



Usually tools like this work with context-free grammar and top-down incremental parser. The problem with this is that the context-free grammars cannot represent well all naturall languages. Even if this somehow works for English, it becomes hopeless for languages with more complicated structure. The parser that we use works directly with full GF and can therefore successfully handle long distance relations and discontinuous phrases.

4 Results

Writing the GF implementation was quick and mostly smooth. The main problem was that some ACE structures are not quite correct in standard English, and therefore not covered by the library. One example is negated noun phrases, especially in the object position: *John meets not every customer*. Porting the grammar to other languages was made in the simplest possible way, by using a parametrized module. The result is mostly acceptable: grammatically correct but sometimes strange, especially for constructs like *not every*. This could be improved by making the concrete syntaxes more unlike Attempto English than we did. On the other hand, since Attempto French, German, and Swedish are artificial languages anyway, such artificialities might well be tolerable in them, as long as they can be understood without training.

The Attempto project has a system for parsing Attempto English, interpreting it semantically, and generating it from formal descriptions such as ontologies. Similar facilities can been defined in GF as well. However, in the case of Attempto this is not necessary: we can quite as well rely on Attempto tools and use GF just as a translation and authoring tool. Then we can communicate with Attempto tools by using Attempto English (which is unambiguous): translate Attempto French, German, and Swedish to English by using GF, and continue processing with Attempto tools. In the other direction, we can generate Attempto English from logical descriptions using Attempto tools, and then translate the results to French, German, and Swedish with GF.

References

Angelov, K. (2009). Incremental Parsing with Parallel Multiple Context-Free Grammars. In *EACL, Athens, Greece*, to appear.

Attempto (2008). Attempto Project Homepage. `attempto.ifi.uzh.ch/site/`.

Bringert, B., K. Angelov, and A. Ranta (2009). Grammatical Framework Web Service, System demo. In *EACL, Athens, Greece*, to appear.

Caprotti, O. (2006). WebALT! Deliver Mathematics Everywhere. In *Proceedings of SITE 2006. Orlando March 20-24*.

Dannélls, D. (2008). Generating Tailored Texts for Museum Exhibits. In *Proceedings of the 6th edition of LREC. The 2nd Workshop on Language Technology for Cultural Heritage (LaTeCH 2008), Marrakech, Morocco*, pp. 17–20.

Dymetman, M., V. Lux, and A. Ranta (2000). XML and multilingual document authoring: Convergent trends. In *COLING, Saarbrücken, Germany*, pp. 243–249.

Hähnle, R., K. Johannisson, and A. Ranta (2002). An Authoring Tool for Informal and Formal Requirements Specifications. In R. D. Kutsche and H. Weber (Eds.), *Fundamental Approaches to Software Engineering*, Vol. 2306 of *LNCS*, pp. 233–248. Springer.

Hallgren, T. and A. Ranta (2000). An extensible proof text editor. In M. Parigot and A. Voronkov (Eds.), *LPAR-2000*, Vol. 1955 of *LNCS/LNAI*, pp. 70–84. Springer.

Larsson, S. and P. Ljunglöf (2008). A grammar formalism for specifying ISU-based dialogue systems. In B. Nordström and A. Ranta (Eds.), *Advances in Natural Language Processing (GoTAL 2008)*, Vol. 5221 of *LNCS/LNAI*, pp. 303–314.

Lemon, O. and X. Liu (2006). DUDE: a Dialogue and Understanding Development Environment, mapping Business Process Models to Information State Update dialogue systems. In *EACL 2006*.

Meza Moreno, M. S. and B. Bringert (2008). Interactive Multilingual Web Applications with Grammarical Framework. In B. Nordström and A. Ranta (Eds.), *Advances in Natural Language Processing (GoTAL 2008)*, Vol. 5221 of *LNCS/LNAI*, pp. 336–347.

Ranta, A. (2004). 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. and R. Cooper (2004). Dialogue Systems as Proof Editors. *Journal of Logic, Language and Information 13*, 225–240.