Grammatical Framework Web Service

Björn Bringert[*] and **Krasimir Angelov** and **Aarne Ranta**
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
{bringert,krasimir,aarne}@chalmers.se

Abstract

We present a web service for natural language parsing, prediction, generation, and translation using grammars in Portable Grammar Format (PGF), the target format of the Grammatical Framework (GF) grammar compiler. The web service implementation is open source, works with any PGF grammar, and with any web server that supports FastCGI. The service exposes a simple interface which makes it possible to use it for interactive natural language web applications. We describe the functionality and interface of the web service, and demonstrate several applications built on top of it.

1 Introduction

Current web applications often consist of JavaScript code that runs in the user's web browser, with server-side code that does the heavy lifting. We present a web service for natural language processing with Portable Grammar Format (PGF, Angelov et al., 2008) grammars, which can be used to build interactive natural language web applications. PGF is the back-end format to which Grammatical Framework (GF, Ranta, 2004) grammars are compiled. PGF has been designed to allow efficient implementations.

The web service has a simple API based solely on HTTP GET requests. It returns responses in JavaScript Object Notation (JSON, Crockford, 2006). The server-side program is distributed as part of the GF software distribution, under the GNU General Public License (GPL). The program is generic, in the sense that it can be used with any PGF grammar without any modification of the program.

2 Grammatical Framework

Grammatical Framework (GF, Ranta, 2004) is a type-theoretical grammar formalism. A GF grammar consists of an *abstract syntax*, which defines a set of abstract syntax trees, and one or more *concrete syntaxes*, which define how abstract syntax trees are mapped to (and from) strings. The process of producing a string (or, more generally, a feature structure) from an abstract syntax tree is called *linearization*. The opposite, producing an abstract syntax tree (or several, if the grammar is ambiguous) from a string is called *parsing*.

In a small, semantically oriented application grammar, the sentence *"2 is even"* may correspond to the abstract syntax tree Even 2. In a larger, more syntactically oriented grammar, in this case the English GF resource grammar (Ranta, 2007), the same sentence can correspond to the abstract syntax tree PhrUtt NoPConj (UttS (UseCl (TTAnt TPres ASimul) PPos (PredVP (UsePN (NumPN (NumDigits (IDig D_2)))) (UseComp (CompAP (PositA *even_A*)))))) NoVoc.

2.1 Portable Grammar Format (PGF)

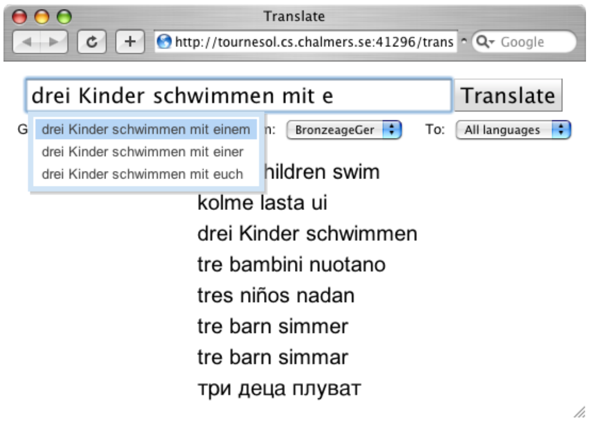
Portable Grammar Format (PGF, Angelov et al., 2008) is a low-level format to which GF grammars are compiled. The PGF Web Service loads PGF files from disk, and uses them to serve client requests. These PGF files are normally produced by compiling GF grammars, but they could also be produced by other means, for example by a compiler from another grammar formalism. Such compilers currently exist for context-free grammars in BNF and EBNF formats, though they compile via GF.

2.2 Parsing and Word Prediction

For each concrete syntax in a PGF file, there is a parsing grammar, which is a Parallel Multiple Context Free Grammar (PMCFG, Seki et al., 1991). The PGF interpreter uses an efficient parsing algorithm for PMCFG (Angelov, 2009) which is similar to the Earley algorithm for CFG. The algorithm is top-down and incremental which makes it possible to use it for word completion. When the whole sentence is known, the parser just takes the tokens one by one and computes the chart of all possible parse trees. If the sentence is not yet complete, then the known tokens can be used to compute a partial parse chart. Since the algorithm is top-down it is possible to predict the set of valid next tokens by using just the partial chart.

The prediction can be used in applications to guide the user to stay within the coverage of the grammar. At each point the set of valid next tokens is shown and the user can select one of them.

[*] Now at Google Inc.



drei Kinder schwimmen mit e

drei Kinder schwimmen mit einem
drei Kinder schwimmen mit einer
drei Kinder schwimmen mit euch

hildren swim
kolme lasta ui
drei Kinder schwimmen
tre bambini nuotano
tres niños nadan
tre barn simmer
tre barn simmar
три деца плуват

Figure 1: Translator interface. This example uses the Bronzeage grammar, which consists of simple syntactic rules along with lexica based on Swadesh lists. Demo at `http://digitalgrammars.com/translate`.

The word prediction is based entirely on the grammar and not on any additional n-gram model. This means that it works with any PGF grammar and no extra work is needed. In addition it works well even with long distance dependencies. For example if the subject is in a particular gender and the verb requires gender agreement, then the the correct form is predicted, independently on how far the verb is from the subject.

3 Applications

Several interactive web applications have been built with the PGF Web Service. They are all JavaScript programs which run in the user's web browser and send asynchronous HTTP requests to the PGF Web Service.

3.1 Translator

The simplest application (see Figure 1) presents the user with a text field for input, and drop-down boxes for selecting the grammar and language to use. For every change in the text field, the application asks the PGF Web Service for a number of possible completions of the input, and displays them below the text field. The user can continue typing, or select one of the suggestions. When the current input can be parsed completely, the input is translated to all available languages.

3.2 Fridge Poetry

The second application is similar in functionality to the first, but it presents a different user interface. The interface (see Figure 2) mimics the popular refrigerator magnet poetry sets. However, in contrast to physical fridge magnets, this application handles inflection automatically and only allows the construction of grammatically correct sentences (as defined by the selected grammar). It also shows translations for complete inputs and allows the user to switch languages.

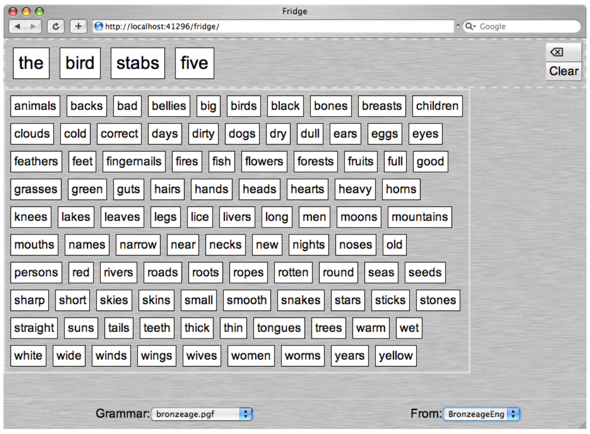
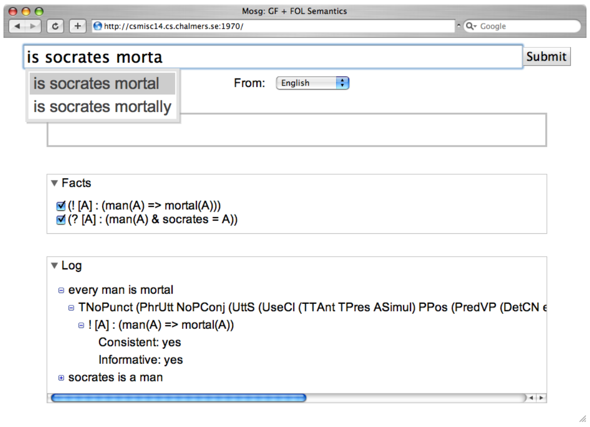

Fridge
http://localhost:41296/fridge/
Google
the
bird
stabs
five
Clear
animals backs bad bellies big birds black bones breasts children
clouds cold correct days dirty dogs dry dull ears eggs eyes
feathers feet fingernails fires fish flowers forests fruits full good
grasses green guts hairs hands heads hearts heavy horns
knees lakes leaves legs lice livers long men moons mountains
mouths names narrow near necks new nights noses old
persons red rivers roads roots ropes rotten round seas seeds
sharp short skies skins small smooth snakes stars sticks stones
straight suns tails teeth thick thin tongues trees warm wet
white wide winds wings wives women worms years yellow
Grammar: bronzeage.pgf
From: BronzeageEng

Figure 2: Fridge poetry screenshot. Demo at `http://digitalgrammars.com/fridge`.



is socrates morta

is socrates mortal
is socrates mortally

▼ Facts

☑ (! [A] : (man(A) => mortal(A)))
☑ (? [A] : (man(A) & socrates = A))

▼ Log

```
⊖ every man is mortal
  ⊖ TNoPunct (PhrUtt NoPConj (UttS (UseCl (TTAnt TPres ASimul) PPos (PredVP (DetCN e
    ⊖ ! [A] : (man(A) => mortal(A))
        Consistent: yes
        Informative: yes
⊕ socrates is a man
```

Figure 3: Reasoning screenshot. Demo at `http://digitalgrammars.com/mosg`.

3.3 Reasoning

Another application is a natural language reasoning system which accepts facts and questions from the users, and tries to answer the questions based on the facts given. The application uses the PGF Web Service to parse inputs. It uses two other web services for semantic interpretation and reasoning, respectively. The semantic interpretation service uses a continuation-based compositional mapping of abstract syntax terms to first-order logic formulas (Bringert, 2008). The reasoning service is a thin layer on top of the Equinox theorem prover and the Paradox model finder (Claessen and Sörensson, 2003).

4 API

Below, we will show URI paths for each function, for example `/pgf/food.pgf/parse`. Arguments to each function are given in the URL query string, in `application/x-www-form-urlencoded` (Raggett et al., 1999) format. Thus, if the service is running on `example.com`, the URI for a request to parse the string "this fish is fresh" using the `FoodEng` concrete syntax in the `food.pgf` grammar would be: `http://example.com/pgf/food.pgf/parse?input=this+fish+is+fresh&from=FoodEng`. The functions described below each accept some subset of the following arguments:

`from` The name of the concrete syntax to parse with or translate from. Multiple `from` arguments can be given, in which case all the specified languages are tried. If omitted, all languages (that can be used for parsing) are used.

`cat` The name of the abstract syntax category to parse or translate in, or generate output in. If omitted, the start category specified in the PGF file is used.

`to` The name of the concrete syntax to linearize or translate to. Multiple `to` arguments can be given, in which case all the specified languages are used. If omitted, results for all languages are returned.

`input` The text to parse, complete or translate. If omitted, the empty string is used.

`tree` The abstract syntax tree to linearize.

`limit` The maximum number of results to return.

All results are returned in UTF-8 encoded JSON or JSONP format. A `jsonp` argument can be given to each function to invoke a callback function when the response is evaluated in a JavaScript interpreter. This makes it possible to circumvent the Same Origin Policy in the web browser and call the PGF Web Service from applications loaded from another server.

4.1 Grammar List

`/pgf` retrieves a list of the available PGF files.

4.2 Grammar Info

`/pgf/grammar.pgf`, where `grammar.pgf` is the name of a PGF file on the server, retrieves information about the given grammar. This information includes the name of the abstract syntax, the categories in the abstract syntax, and the list of concrete syntaxes.

4.3 Parsing

`/pgf/grammar.pgf/parse` parses an input string and returns a number of abstract syntax trees. Optional arguments: `input`, `from`, `cat`.

4.4 Completion

`/pgf/grammar.pgf/complete` returns a list of predictions for the next token, given a partial input. Optional arguments: `input`, `from`, `cat`, `limit`. If `limit` is omitted, all results are returned.

4.5 Linearization

`/pgf/grammar.pgf/linearize` accepts an abstract syntax tree, and returns the results of linearizing it to one or more languages. Mandatory arguments: `tree`. Optional arguments: `to`.

4.6 Random Generation

`/pgf/grammar.pgf/random` generates a number of randomly generated abstract syntax trees for the selected grammar. Optional arguments: `cat`, `limit`. If `limit` is omitted, one tree is returned.

4.7 Translation

`/pgf/grammar.pgf/translate` performs text to text translation. This is done by parsing, followed by linearization. Optional arguments: `input`, `from`, `cat`, `to`.

5 Application to Controlled Languages

The use of controlled languages is becoming more popular with the development of Web and Semantic Web technologies. Related projects include Attempto (Attempto, 2008), CLOnE (Funk et al., 2007), and Common Logic Controlled English (CLCE) (Sowa, 2004). All these projects provide languages which are subsets of English and have semantic translations into first order logic (CLCE), OWL (CLOnE) or both (Attempto). In the case of Attempto, the translation is into first order logic and if it is possible to the weaker OWL language.

The general idea is that since the controlled language is a subset of some other language it should be understandable to everyone without special training. The opposite is not true - not every English sentence is a valid sentence in the controlled language and the user must learn how to stay within its limitations. Although this is a disadvantage, in practice it is much easier to remember some subset of English phrases rather than to learn a whole new formal language. Word suggestion functionality such as that in the PGF Web Service can help the user stay within the controlled fragment.

In contrast to the above mentioned systems, GF is not a system which provides only one controlled language, but a framework within which the developer can develop his own language. The task is simplified by the existence of a resource grammar library (Ranta, 2007) which takes care of all low-level details such as word order, and gender, number or case agreement. In fact, the language developer does not have to be skilled in linguistics, but does have to be a domain expert and can concentrate on the specific task.

Most controlled language frameworks are focused on some subset of English while other languages receive very little or no attention. With GF, the controlled language does not have to be committed to only one natural language but could have a parallel grammar with realizations into many languages. In this case the user could choose whether to use the English version or, for example, the French version, and still produce the same abstract representation.

6 Implementation

The PGF Web Service is a FastCGI program written in Haskell. The program is a thin layer on top of the PGF interpreter, which implements all the PGF functionality, such as parsing, completion and linearization. The web service also uses external libraries for FastCGI communication, and JSON and UTF-8 encoding and decoding.

The main advantage of using FastCGI instead of plain CGI is that the PGF file does not have to be reloaded for each request. Instead, each PGF file is loaded the first time it is requested, and after that, it is only reloaded if the file on disk is changed.

7 Performance

The web service layer introduces minimal overhead. The typical response time for a parse request with a small grammar, when running on a typical current PC, is around 1 millisecond. For large grammars, response times can be on the order of several seconds, but this is entirely dependent on the PGF interpreter implementation.

The server is multi-threaded, with one lightweight thread for each client request. A single instance of the server can run threads on all cores of a multi-core processor. Since the server maintains no state and requires no synchronization, it can be easily replicated on multiple machines with load balancing. Since all requests are cacheable HTTP GET requests, a caching proxy could be used to improve performance if it is expected that there will be repeated requests for the same URI.

8 Future Work

The abstract syntax in GF is based on Martin Löf's (1984) type theory and supports dependent types. They can be used go beyond the pure syntax and to check the sentences for semantic consistency. The current parser completely ignores dependent types. This means that the word prediction will suggest completions which might not be semantically meaningful.

In order to improve performance for high-traffic applications that use large grammars, the web service could cache responses. As long as the grammar is not modified, identical requests will always produce identical responses.

9 Conclusions

We have presented a web service for grammar-based natural language processing, which can be used to build interactive natural language web applications. The web service has a simple API, based on HTTP GET requests with JSON responses. The service allows high levels of performance and scalability, and has been used to build several applications.

References

Krasimir Angelov. 2009. Incremental Parsing with Parallel Multiple Context-Free Grammars. In *European Chapter of the Association for Computational Linguistics*.

Krasimir Angelov, Björn Bringert, and Aarne Ranta. 2008. PGF: A Portable Run-Time Format for Type-Theoretical Grammars. *Journal of Logic, Language and Information*, submitted. URL `http://www.cs.chalmers.se/~bringert/publ/pgf/pgf.pdf`.

Attempto. 2008. Attempto Project Homepage - http://attempto.ifi.uzh.ch/site/. URL `http://attempto.ifi.uzh.ch/site/`.

Björn Bringert. 2008. Delimited Continuations, Applicative Functors and Natural Language Semantics. URL `http://www.cs.chalmers.se/~bringert/publ/continuation-semantics/continuation-semantics.pdf`.

Koen Claessen and Niklas Sörensson. 2003. New Techniques that Improve MACE-style Model Finding. In *Workshop on Model Computation (MODEL)*. URL `http://www.cs.chalmers.se/~koen/pubs/model-paradox.ps`.

Douglas Crockford. 2006. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational). URL `http://www.ietf.org/rfc/rfc4627.txt`.

Adam Funk, Valentin Tablan, Kalina Bontcheva, Hamish Cunningham, Brian Davis, and Siegfried Handschuh. 2007. CLOnE: Controlled Language for Ontology Editing. In *Proceedings of the International Semantic Web Conference (ISWC 2007)*. Busan, Korea.

Per Martin-Löf. 1984. *Intuitionistic Type Theory*. Bibliopolis, Naples.

Dave Raggett, Arnaud Le Hors, and Ian Jacobs. 1999. HTML 4.01 Specification. Technical report, W3C. URL `http://www.w3.org/TR/1999/REC-html401-19991224/`.

Aarne Ranta. 2004. Grammatical Framework: A Type-Theoretical Grammar Formalism. *Journal of Functional Programming*, 14(2):145–189. URL `http://dx.doi.org/10.1017/S0956796803004738`.

Aarne Ranta. 2007. Modular Grammar Engineering in GF. *Research on Language and Computation*, 5(2):133–158. URL `http://dx.doi.org/10.1007/s11168-007-9030-6`.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229. URL `http://dx.doi.org/10.1016/0304-3975(91)90374-B`.

John Sowa. 2004. Common Logic Controlled English. Draft. URL `http://www.jfsowa.com/clce/specs.htm`.