The GF Mathematical Grammar Library: from OpenMath to natural languages

Olga Caprotti[1] and Jordi Saludes[2] *

[1] Chalmers and University of Gothenburg
Sweden
caprotti@chalmers.se
[2] Dpt. Matemàtica Aplicada 2,
Universitat Politècnica de Catalunya
08201 Terrassa, Spain
jordi.saludes@upc.edu

1 Introduction

Since 2005 we have been developing a software library for rendering, reading, and translating mathematical expressions, either expressed using formal languages such as OpenMath and LaTeX, or in multiple natural languages. The work started with the WebALT [1] project as a way to serve mathematical exercises in the native language of the student: in fact the library can be used to generate natural language descriptions of formally encoded mathematical expressions with no loss of meaning. The applications of this technology, coming from the area of grammar-based machine translation are related to the possibility of parsing and generating high quality representations of mathematics.

In this paper we want to concentrate on the technical details that made the work interesting from the linguistic point of view. Therefore we introduce the computational linguistic software that was used as backbone to the work, called Grammatical Framework, and proceed with the presentation of the mathematical library, its organization and modular design. We then discuss some examples that required careful thought.

1.1 The Grammatical Framework

To deal with multilingualism, the chosen tool was the *Grammatical Framework* (GF): a type theoretic programming language for writing grammars for multiple languages at once [3]. This is done using an interlingua: the semantics of an expression in natural language that should be rendered or translated is captured in an *abstract tree*, which is its language-independent representation. The so-called *abstract grammar* determines what is possible to express in the specific application, whilst the *concrete grammars* (one for each language) define how the abstract meaning is converted to the given language. Once an abstract grammar is given, to add yet another language to the application amounts to adding a new concrete grammar. Ideally, if a concrete grammar for a language in the same linguistic group is already available, the grammar for the new language is almost an exact copy of the existing grammar, modulo some lexicon adaptations. GF hides all linguistic details of a specific language from the programmer in a low-level *resource grammar library*, so you need only a domain expert to develop new languages

*The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° FP7-ICT-247914.

for a given application, a language expert is not needed. Details of the GF Grammar Library, including language coverage, are online[1].

A GF abstract grammar defines how expressions in given *categories* are combined. This is how an example tree in the mathematical grammar library looks like:

```
mkProp
  (lt_num (abs (plus (BaseValNum (Var2Num x) (Var2Num y))))
  (plus (BaseValNum (abs (Var2Num x)) (abs (Var2Num y)))))
```

When linearized with the English and Spanish concrete grammars, it yields [2]:

> the absolute value of the sum of x and y is less than the sum of the absolute value of x and the absolute value of y
>
> el valor absoluto de la suma de x e y es menor que la suma del valor absoluto de x y el valor absoluto de y

OpenMath experts will notice that the abstract tree is not far from the OpenMath expression. The linguistic function `mkProp` wraps recursively the wording produced by the subexpressions.

The number of categories on a GF application is a trade-off between how much ambiguity is tolerable and the expressiveness of the whole system. In the present case the defined categories are `Value` X, and `Variable` X where X is a `Number`, a `Function`, a `Set` or a `Tensor` (namely vectors or matrices). The actual version of the library, implements these by defining a fixed category for each combination $\{\text{Variable}, \text{Value}\} \times \{\text{Number}, \text{Set}, \text{Function}, \text{Tensor}\}$. Thus, for instance, `VarNum` = `Variable Number` and `ValSet` = `Value Set`. Other categories stand for propositions, geometric constructions and indexes.

These abstract categories correspond to linguistic categories when dealing with the *concrete grammar* of a specific language. Usually a `Value` points to a *noun phrase* and a `Variable` to a string. More complex expressions that combine these categories correspond in a natural way to linguistic entities composed from these elements: propositions are mapped into *clauses* with grammatical polarity, `Operation`s to *sentences* and simple exercises to *texts*.

2 The library

The library can be organized in a matrix, where the horizontal axis runs over the languages, which are at present: Bulgarian, Catalan, English, Finnish, French, German, Hindi, Italian, Polish, Romanian, Russian, Spanish, Swedish and Urdu. It includes also a couple of man-made languages which are relevant to mathematics: LaTeX and Sage [2].

The vertical axis runs over three layers of increasing complexity:

1. **Ground**: literals, indexes and variables
2. **OpenMath**: modeled after the following *Content Dictionaries*, considered useful for expressing the mathematical fragments at the time of the WebALT project:
   - arith1, arith2, complex1, integer1, integer2, logic1, nums1, quant1, relation1, rounding1;

[1] http://www.grammaticalframework.org/lib/doc/synopsis.html

[2] Notice the special form of the conjunction "x e y": The usual Spanish conjunction "y" must be changed for euphony before a vowel that sounds alike. It is automatically taken care by the GF Spanish resource grammar.

- calculus1, fns1, fns2, interval1, limit1, transc1, veccalc1;
- linalg1, linalg2;
- minmax1, plangeo1, s_data1, set1, setname1.

3. **Operations**: takes care of simple mathematical exercises. These appear in drilling exercises and usually begin with directives such as 'Compute', 'Find', 'Prove', 'Give an example of', etc.

Objects in the OpenMath standard [5] relate to GF types, namely each symbol in a Content Dictionary (CD) roughly corresponds to a production of the same name in a GF module named after that CD. Application of functions to numbers are expressed by the production `At` that takes a `Value Function` and a `Value Number` and return a `Value Number`. More examples are in the table 1.

Following the lines of the *Small Type System* [4, principle 4], we imposed that binary associative functions take a list of values and return a value of the same kind. For example, `plus` in `arith1` has signature plus : [ValNum] $\to$ ValNum, while the category [ValNum] (meaning a list of numeric values) is declared to take at least two values. Therefore is impossible by construction to add a single number (i. e. "the sum of 3").

3 Linguistic peculiarities

Some interesting points on the implementation are related to language specifics. For example, the simple exercise that asks for computing a numeric value [3]:

```
DoComputeN ComputeV  (determinant (Var2Tensor M))
```

gives in English:

Compute the determinant of M.

This pattern is shared in most of the languages, so it got abstracted into an *incomplete concrete grammar* file **OperationsI**. From this module, one can get **Operations**L for language L simply by specifiying the lexicon and paradigms modules for this L, in a similar way a function is applied to its arguments. But in French is impolite to use an imperative in this case; Therefore the module **OperationsFre** should re-implement this production in a specific way.

Another point worth mentioning is *function application.* Notice the different forms:

- "the cosine $\underline{\text{of}}$ 3"
- "f $\underline{\text{at}}$ 3"
- "the derivative of the sine $\underline{\text{at}}$ 3"
- "x to the cosine of x $\underline{\text{where } x \text{ is}}$ 3"

They are all mathematically equivalent but differ in structure: in the first case, the function being applied is a named symbol (the cosine) while in the last one is a λ-abstraction. In the other cases, it is a function variable or it comes from a functional operator.

[3] `determinant` belongs to the OpenMath layer of the library and `Var2Tensor` makes a value out of a variable. `DoComputeN` denotes an exercise asking to compute a number, while `ComputeV` gives finer control on which verb to use to denote computation ('to compute' in this case).

OpenMath	GF
Symbol ***name*** in ***CD***	*`name`* in module *`CD`*
Integer n	n converted to `Value` from predefined type `Int` in module `Literals`
Variable ***name***	*`name`* in category `Variable` X
Application of a on b	a b
Binding λz *app*	`lambda` z *app*, where z is a `Variable` and *app* a `Value`.
Attribution, Error, Bytearray	Not supported

Table 1: Some equivalences between the OpenMath standard and grammar library

4 Applications and future development

The library is publicly available at the MOLTO repository [7] and is documented at [8]. It is being used in the *mathbar* demo in the MOLTO project [6] accessible from [9]. An example of natural language interaction with a computer algebra system can be retrieved from the `sage` directory of the library distribution and has been recently presented at [12].

For the future, the library needs to grow in breadth and shape: at this moment, it is systematically tested for three languages but depends on domain experts native speakers to polish the remaining ones.

Integration of natural language productions and formulas is also prominent in the TODO list. This is a variegated issue as [10] shows, but it is necessary for fluent mathematics in applications. Also more natural renderings of logical propositions [11] will open the door to usage in automatic reasoners and theorem provers.

References

[1] `http://webalt.math.helsinki.fi/content/index_eng.html` Last viewed June 2012.

[2] `http://www.sagemath.org/` Last viewed May 2012.

[3] A. Ranta, "Grammatical Framework: programming with Multilingual Grammars," CSLI Studies in Computational Linguistics, Standford, 2011.

[4] J. H. Davenport, "A small OpenMath type system," ACM SIGSAM Bulletin, vol. 34, no. 2, pp. 16–21, Jun. 2000.

[5] OpenMath Consortium, "The OpenMath Standard," OpenMath Deliverable, vol. 1, 2000.

[6] The MOLTO project. `http://www.molto-project.eu/` Last viewed May 2012.

[7] The mathematical library `svn://molto-project.eu/mgl`.

[8] J. Saludes et al., "Simple drill grammar library," `http://www.molto-project.eu/sites/default/files/d61.pdf`. Last viewed May 2012.

[9] Grammatical framework demos. `http://www.grammaticalframework.org/demos/index.html`. Last viewed May 2012.

[10] Mohan Ganesalingam, "The Language of Mathematics." PhD thesis, Cambridge University, 2009.

[11] A. Ranta, "Translating between language and logic: what is easy and what is difficult." Automated Deduction, CADE-23, 2011.

[12] Dominique Archambault, Olga Caprotti, Aarne Ranta and Jordi Saludes, "Using GF in multimodal assistants for mathematics." Digitization and E-Inclusion in Mathematics and Science 2012, Tokyo, Japan.