TOWARD MULTILINGUAL MECHANIZED MATHEMATICS ASSISTANTS*

JORDI SALUDES AND SEBASTIAN XAMBÓ

Abstract. The aim of this note is to present recent work in the area of Multilingual Mechanized Mathematics Assistants. We will describe some of the key features expected from such systems, sketch some of the strategies envisioned for their realization and report on the current state of the Mathematical Grammar Library.

Introduction

Computational methods have become ever more important in the last decades. They are used to solve problems in a great variety of research and applied fields, as underlined by EACA's list of topics and by similar lists of related conferences. As computational speed and algorithmic sophistication increase, new areas that not so long ago were judged to be out of reach appear to be amenable to a computational approach. Often, these new approaches are multidisciplinary by nature, if only because they must harness and profit from computational expertise in different fields.

One endeavour that is strengthening its reliance on computational approaches aims at the design and implementation of multilingual mechanised mathematics assistants (MMMA). The purpose of this note is to present some of the key features expected from such systems and some of the strategies envisioned for their realization. More specifically, we will have a closer look on multilingual mathematical dialog systems designed to teach students to solve and how to solve *word problems* at the level of high school and freshmen linear algebra and calculus.

1. Word Problems

They play a prominent role in the learning of mathematical modelling and reasoning [1, 2]. Let us illustrate the idea with a simple example:

> In a family party, uncle Tom asks Alice how old she is. The day before yesterday I was 15, said Alice, and next year I will be 18. What day is Alice's birthday?

In these kind of problems, most of the relevant information is given as a natural language text. The translation of sentences into mathematical expressions and the reasoning with them, including the checking of the solution, are the crucial skills to be learned. This learning is usually hard, basically because key facts and relations are not explicitly indicated in the

* The research leading to the results presented here has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-247914.

problem statement. Actually, facts and relations have to be digged out by using common knowledge about the terms. An additional difficulty is sorting out which facts and relations have to be taken into account for the solution from those that are irrelevant.

In our example these difficulties can be highlited by relating the statement to the solution, which may be outlined as follows:

> If Alice will be 18 next year, this year she is or will be 17.
> If the day before yesterday she was 15, her 16th birthday must have been yesterday.
> Thus yesterday and today must be in two consecutive years.
> The only possibility is that her birthday falls on 31st December.

We see that the solution relies on having a clear grasp of terms such as 'age', 'yesterday', 'day before', 'year', 'this year' or 'next year', together with the tacit relations among them that warrant the statements in the solution, whereas terms like 'familiy', 'family party', 'uncle' or 'uncle Tom' do not play any role.

2. Toward multilingual mathematics

Our work on multilingual processing of mathematical text began in the context of the WebALT project[1] that aimed at using the then existing standards for representing mathematics on the web and existing linguistic technologies to produce language-independent mathematical didactical material [3, 4, 5, 6].

This development has continued with the mOlto project (see footnote on the first page) and at present it is focussed on the development of MGL, the Mathematical Grammar Library. It is a modular library programmed in GF[2] in a way that is comparable to how numerical libraries are compiled from C or Fortran sources. The general purpose of MGL is to parse multilingual mathematical text with mathematical expressions into an abstract formal encoding (formal semantics) and translate this encoding into other languages [9].[3] At present it can deal with rather simple exercises (see, for example, the demo [10]) in thirteen natural languages (plus LaTeX).[4]

3. Ancillary CAS and ATP systems

Since the formal semantics produced by MGL can be processed algorithmically, it actually can be the basis for powerful interactions with ancillary Computer Algebra Systems (CAS) and Automated Theorem Provers (ATP).

Managing calls to an ancillary ATP is a necessary feature of any advanced MMA, as it is a required element in the process of automatically assembling a solution of the problem

[1] European digital content for the global networks project (2005 and 2006, contract Number EDC-22253).

[2] Grammatical Framework. It is a programming language for multilingual grammar applications. Based on functional programming and type theory, the framework supports abstract grammars, which allow to capture meaning in a formal way, and concrete grammars, which enable correct multilingual rendering. For a systematic presentation, see [7]. For a quicker introduction, see [8].

[3] The living end of the library is publicly available using subversion as

```
svn co svn://molto-project.eu/mgl.
```

A stable version can be found at

```
svn co svn://molto-project.eu/tags/D6.1.
```

[4] Bulgarian, Catalan, English, Finnish, French, German, Italian, Polish, Romanian, Russian, Spanish, Swedish and Urdu.

and for checking the users input statements. Indeed, ATPs are computer programs capable of supplying proofs of mathematical theorems. In other words, they purport to show that some (conjectural) statement can be logically derived from a set of axioms and hypotheses. They can be used in a great variety of situations, including of course mathematics. They belong to the more general field of Automated Reasoning (AR) and usually they support some degree of interactivity, in the sense that some parts of the proofs, or some options, may have to be input by the user.[5]

It is worthwhile to note that an ATP system can only be used on precise formal statements written according to their native grammar. This is a condition that is far from being satisfied by customary word problems, even if they were phrased in a controlled language such as Attempto's English. Hence one of the main strengths of MGL is that it is a good start toward mechanizing the logical formalization of word problems out of their natural language expression.

Another key aspect of an advanced MMA is the capability of managing calls to an ancillary CAS system. This is a necessary feature of any advanced MMA, as it is required to check whether mathematical expressions input by users satisfy the required constraints. In fact, it is also required to perform computations called by the MMA along the process of building a reasoned solution. The prototype presented in next section indicates how Sage might be used as an ancillary CAS system in a multilingual setting.

Sage commands in natural language

A recently developed prototype based on MGL is `gfsage`. It enables to express Sage[6] commands in natural language and get the results phrased likewise. The tool starts a Sage session in the background (as described in Simple Sage Server API, [12]), reads the pgf grammar file and translates the queries from the chosen natural language to the concrete grammar for Sage. This is passed to the Sage server for evaluation and the server replies with a `done` or a `computing` message. In this case the program waits for completion of the computation and then writes the answer.

From the GF side, what is send to Sage is always in the category `Command`. What is returned by Sage is in the category `Answer`.[7] There are 3 kinds of Commands:

- Asking for a computation. Compute: `Kind -> Value Kind -> Command`.
  Sage gives back a `ReturnBlock` with the cell number and the answer (a string).
  We could now construct a short `Answer` by using:
  - Simple: `k ∈ Kind -> Value k -> Answer`
    ("it is 5"), or

[5] Links to some ATP: Prover9 and Mace4, E, SPASS, ACL2, HOL, Isabelle Coq, EQP, Vampire, Waldmeister. See Geoff Sutcliffe's Overview of ATP, and [11].

[6] Aimed at "creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab", Sage is the result of an on-going collective endeavour led by William Stein.

[7] Any GF application begins by specifying its *abstract syntax*. This syntax contains declarations of *categories* (the GF name for types) and *functions* (the GF name for constructor signatures) and has to capture the *semantic structure* of the application domain.

  - Feedback: `k ∈ Kind -> Value k -> Value k -> Answer`
    ("the factorial of 3 is 6"), that combines the question (the first Value k) with the Sage answer.
- Assuming propositions. Assume: `Prop -> Command`.
  Sage silently accepts the command by returning an `EmptyBlock` (with cell number) but we want it to be more assertive, so we reinject the `Prop` into Assumed: `Prop -> Answer`
  ("I assume that x is greater than 2")
- Binding Values to Variables.
  Assign: `k ∈ Kind -> Var k -> Value k -> Command`
  ("assign 2 to x").
  We expect Sage to return an `EmptyBlock` followed by
  Assigned: `k ∈ Kind -> Var k -> Value k -> Answer`
  ("2 is now assigned to x").

Example:

```
sage> compute the integral of the function mapping x
      to the square root of x on the closed interval from 1 to 2.
waiting...
[4] 4/3*sqrt(2) - 2/3
answer: it is 4/3*sqrt(2) - 2/3
```

References

[1] L. Verschaffel, B. Greer, and E. De Corte. *Making sense of word problems*. Taylor & Francis, 2000.
[2] D. Wayne. *How to solve word problems in mathematics*. McGraw-Hill, 2001.
[3] O. Caprotti. Webalt! Deliver mathematics everywhere. In *Proceedings of SITE 2006*, 2006.
[4] O. Caprotti and M. Seppälä. Multilingual delivery of online tests in mathematics. In *Proceedings of Online Educa Berlin 2006*, 2006.
[5] S. Xambó, H. Bass, G. Bolanos, R. Seiler, and M. Seppälä. E-learning mathematics. In *Proceedings of the ICM-2006 (Volume III)*, pages 1743–1768. European Mathematical Society, 2006.
[6] S. Xambó, O. Caprotti, and M. Seppälä. Toward autonomous learners of mathematics. In E. M. Rocha J. M. Borwein and J. F. Rodrigues, editors, *Communicating Mathematics in the Age of Digital Libraries*, pages 239–252. A. K. Peters, 2008.
[7] A. Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011.
[8] A. Ranta. Grammatical framework. `http://www.grammaticalframework.org/`.
[9] J. Saludes and S. Xambó. The GF mathematics library. In *Proceedings First Workshop on CTP Components for Educational Software Wrocław, Poland, 31th July 2011*, volume 79, pages 102–110, 2012.
[10] T. Hallgren and J. Saludes. Math Bar Online, 2010. `http://www.grammaticalframework.org/demos/minibar/mathbar.html`.
[11] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.
[12] Sage developing team. Simple Sage Server API, Consulted Jan 15, 2012.

Jordi Saludes: Universitat Politècnica de Catalunya, Edifici GAIA, Terrassa (Spain)
E-mail address: `jordi.saludes@upc.edu`

Sebastian Xambó: Universitat Politècnica de Catalunya, Edifici OMEGA, Barcelona (Spain)
E-mail address: `sebastia.xambo@upc.edu`