New developments in parsing mizar

Czesław Bylinski
Department of Programming and Formal Methods
University of Białystok
Poland
czeslaw@mizar.org

Jesse Alama*
Center for Artificial Intelligence
New University of Lisbon
Portugal
j.alama@fct.unl.pt

Abstract

The mizar language aims to capture mathematical vernacular by providing a rich language for mathematics. From the perspective of a user, the richness of the language is welcome because it makes writing texts more "natural". But for the developer, the richness leads to syntactic complexity, such as dealing with overloading.

As part of the larger project of open-sourcing the mizar codebase, the mizar team is opening up the mizar parser. The result has led them to consider afresh the problems of parsing the mizar language and making it accessible to users and other developers. In this paper we describe these new parsing efforts and some applications thereof, such as large-scale text refactorings, pretty-printing, HTTP parsing services, and normalizations of mizar texts.

1 Introduction

The mizar system provides a language for declaratively expressing mathematical content and writing mathematical proofs. One of the principal aims of the mizar project is to capture "the mathematical vernacular" by permitting authors to use linguistic constructions that mimic ordinary informal mathematical writing. The richness is welcome for authors of mizar texts. However, a rich, flexible, expressive language is good for authors can lead to difficulties for developers and enthusiasts. Certain experiments with the mizar language and its vast library of formalized mathematical knowledge (the mizar Mathematical Library, or MML), naturally lead to rewriting mizar texts in various ways. Until recently, there has been no standalone tool, distributed with mizar, that would simply parse mizar texts and present the parse trees in a workable form.[1] Compounding the problem of the inherent difficulty of parsing mizar texts and the lack of a standalone parser, mizar itself is, to a large extent, a closed-source system.[2] One could try to reverse-engineer mizar to carry out private experiments, but this is a steep obstacle. One can request assistance from the mizar developers for specific tasks[3], but this is obviously an unacceptable method for solving one's own problems. And even if one has access to the mizar codebase, it may not be clear how to carry out a specific rewriting task.

The mizar team is moving toward releasing their code under a standard open-source license. Part of this process is a reorganization of the mizar toolchain, starting with the parser. This paper documents these efforts and illustrates some of the fruits they have already borne. This paper does not explain *how* to parse arbitrary mizar texts; that is a thorny issue of its own. And for lack of space we cannot go into the detail about the mizar system; see [5, 6]. The scope of this paper is to announce new developments by the mizar team that make parsing of mizar texts more accessible for users and developers.

*Supported by the ESF research project *Dialogical Foundations of Semantics* within the ESF Eurocores program *LogICCC* (funded by the Portuguese Science Foundation, FCT LogICCC/0001/2007). Research for this paper was partially done while a visiting fellow at the Isaac Newton Institute for the Mathematical Sciences in the programme 'Semantics & Syntax'.

[1] One parser tool, *lisppars*, is distributed with mizar. *lisppars* is mainly used to facilitate authoring mizar texts with Emacs [7]; it carries out fast lexical analysis only and does not output parse trees.

[2] The contents of the mizar Mathematical Library, on the other hand, are now governed by an open-source license [3].

[3] The first author would like to thank Karol Pąk for his patient assistance in developing customized mizar text rewriting tools.

In Section 2, we discuss different views of mizar texts that are now available. Section 3 describes some current applications made possible by opening up mizar texts, and describes some HTTP-based services for those who wish to connect their own tools to mizar services. Section 4 concludes by sketching further work and potential applications.

2 Layers of a mizar text

It is common in parsing theory to distinguish various analyses of a text [2]. Traditionally the first task in parsing is **lexical analysis** or **scanning**: to compute, from a stream of characters, a stream of *tokens*, i.e., terminals of a production grammar G. From a stream of tokens one then carries out a **syntactic analysis**, which is the synthesis of tokens into groups that match the production rules of G.

One cannot, in general, lexically analyze mizar texts without access to the MML because lexical analysis requires access to the library's store of symbols. Consider, for example, mizar text fragment

```
fx=a
```

consisting four characters. Several lexical analyses of this fragment are possible. If `fx` and `a` are known to be function symbols, then one could analyze this text fragment as the sequence ⟨term⟩⟨equality⟩⟨term⟩ of tokens of length 3. If `fx=a` is a function symbol, then the correct lexical analysis is ⟨term⟩. (This example is legal because although the equality symbol is one of the keywords in the mizar grammar, it is not forbidden to define function symbols whose name includes equality as a part.) Perhaps `fx=` and `a` are function symbols, so the fragment should be lexically analyzed as ⟨term⟩⟨term⟩. If instead `fx` and `=a` are function symbols, then the right analysis is ⟨term⟩⟨term⟩, which is congruent with the previous lexical analysis, but the lexical contents of the tokens differ. If `fx=a` is a relation symbol, then the correct lexical analysis is ⟨relation⟩. Further lexical analyses are possible, but these ought to be enough examples to illustrate the difficulty.

Even with a lexical analysis of a mizar text, how should it be understood syntactically? Consider

```
let X be set,
    R be Relation of X, Y;
```

The difficulty in this case comes from **dependent types**. There is the notion *(binary) relation*, indicated by the non-dependent type `Relation`. There is also the notion *relation whose domain is a subset of X and whose range is a subset of Y*, which is expressed using the dependent type `Relation of X,Y`. Finally, we have the notion *relation whose domain is a subset of X and whose range is a subset of X* which is written `Relation of X`. In the text fragment above, we have to somehow determine which possibility is correct, but this information is not contained in the token stream.

2.1 Normalizations of mizar texts

One goal of opening up the mizar parser is to help those interested in working with mizar texts to not have to rely on the mizar codebase to do their own experiments with mizar texts. We now describe two normalizations of (arbitrary) mizar texts, which we call weakly strict and more strict. The results of these two normalizations on a mizar text can be easily parsed by a standard LR parser, such as those generated by the standard tool *bison*[4] and have further desirable syntactic and semantic properties.

2.2 Weakly strict mizar

The aim of the weakly strict mizar (WSM) transformation is to define a class of mizar texts for which one could easily write an standard, standalone parser that does not require any further use of the mizar

[4] `http://www.gnu.org/software/bison/`

tools. In a weakly strict mizar text all notations are disambiguated and fully parenthesized, and all statements take up exactly one line. (This is a different transformation than single-line variant AUT-SL of the Automath system [4].) Consider:

```
reserve P,R for Relation of X,Y;
```

This mizar fragment is ambiguous: it is possible that the variable `Y` is a third reserved variable (after the variables `P` and `R`), and it is possible that `Y` is an argument of the dependent type `Relation of X,Y`. The text becomes disambiguated by the weakly strict mizar normalization to

```
reserve P , R for ( Relation of X , Y ) ;
```

and now the intended reading is syntactically evident, thanks to explicit bracketing and whitespace.

The result of the one-line approach of the weakly strict mizar normalization is, in many cases, excessive parenthesization, unnecessary whitespace, and rather long lines.[5] The point of the weakly strict mizar normalization is not to produce attractive human-readable texts. Instead, the aim is to transform mizar texts so that they have a simpler grammatical structure.

2.3 More Strict mizar

A second normalization that we have implemented is called, for lack of a better term, more strict mizar (MSM). The aim of the MSM normalization is to to define a class of mizar texts that are canonicalized in the following ways:

- From the name alone of an occurrence of a variable one can determine the category (reserved variable, free variable, bound variable, etc.) to which the occurrence belongs.
- All formulas are labeled, even those that were unlabeled in the original text.
- Some "syntactic sugar" is expanded ("unsweetened").
- Toplevel logical linking is replaced by explicit reference. Thus,

  ϕ`; then` ψ`;`

  using the keyword `then` includes the previous statement (ϕ) as the justification of ψ. Under the MSM transformation, such logical relationships are rewritten as

  `Label1:` ϕ`;`
  `Label2:` ψ `by Label1;`

  Now both formulas have new labels `Label1` and `Label2`. The logical link between ϕ and ψ, previously indicated by the keyword `then`, is replaced by an explicit reference to the new label (`Label1`) for ϕ.
- All labels of formulas and names of variables in a mizar are serially ordered.

MSM mizar texts are useful because they permit certain "semantic" inferences to be made simply by looking at the syntax. For example, since all formulas are labeled and any use of a formula must be done through its label, one can infer simply by looking at labels of formulas in a text whether a formula is used. By looking only at the name of a variable, one can determine whether it was introduced inside the current proof or was defined earlier.

[5] The longest line in the "WSM-ified" library has length 6042. About 60% (to be precise, 694) of the articles in the WSM form of the current version of the mizar Mathematical Library (4.181.1147) have lines of length at least 500 characters. The average line length across the whole "WSM-ified" library is 54.7.

3 Applications

Opening up the mizar parser facilitates further useful text transformations, such as pretty printing. One can design, for example, XSLT stylesheets [1] for operating on the XML output of the new parser tools.

An HTTP parsing service for these new developments is available for public consumption. Four services are available. Submitting a suitable `GET` request to the service and supplying a mizar text in the message body, one can obtain as a response:

- The WSM form of the text.
- The MSM form of the text.
- The XML representation of the parse tree of the WSM form of the text.
- The XML representation of the parse tree of the MSM form of the text.

The HTTP services permit users to parse mizar texts without having access to the MML, or even the mizar tools. See

`http://mizar.cs.ualberta.ca/parsing`

to learn more about the parsing service, how to prepare suitable HTTP parsing requests, and how to interpret the results.

4 Conclusion and Future Work

Parsing is an essential task for any proof assistant. In the case of mizar, parsing is a thorny issue because of the richness of its language and its accompanying library. New tools for parsing mizar, with an eye toward those who wish to design their own mizar applications without (entirely) relying on the mizar tools, are now available. Various normalizations for mizar texts have been defined. Further useful normalizations are possible. At present we are experimenting with a so-called "variable free" mizar (VFM), in which there are no so-called reserved variables; in VFM texts the semantics of any formula is completely determined by the block in which it appears, which should make processing of mizar texts even more efficient.

References

[1] XSL Transformations (XSLT). Technical report, W3C, 1999. Available online at `http://www.w3.org/TR/xslt`.

[2] A.V. Aho, M.S. Lam, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, and Tools*. Pearson/Addison Wesley, 2007.

[3] J. Alama, M. Kohlhase, L. Mamane, A. Naumowicz, P. Rudnicki, and J. Urban. Licensing the Mizar mathematical library. *Intelligent Computer Mathematics*, pages 149–163, 2011.

[4] N. G. de Bruijn. *AUT-SL, a single-line version of Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, chapter B.2, pages 275–281. North-Holland, 1994.

[5] A. Grabowski, A. Kornilowicz, and A. Naumowicz. Mizar in a nutshell. *Journal of Formalized Reasoning*, 3(2):153–245, 2010.

[6] R. Matuszewski and P. Rudnicki. Mizar: the first 30 years. *Mechanized Mathematics and its Applications*, 4(1):3–24, 2005.

[7] J. Urban. MizarMode—an integrated proof assistance tool for the Mizar way of formalizing mathematics. *Journal of Applied Logic*, 4(4):414 – 427, 2006.