Using and Parsing the Mizar Language

Paul Cairns & Jeremy Gow
UCL Interaction Centre, University College London
31–32 Alfred Place, London WC1E 7DP, UK
{p.cairns,j.gow}@ucl.ac.uk

Abstract

Mizar is a well established and successful system for producing formal mathematics. We investigate the acceptability of formal mathematics to mathematicians by studying the Mizar language. Specifically, we analyse various features of the Mizar language through the exercise of trying to build a grammar for it in order to parse its library. Our analysis highlights unresolved problems with the language which may have reduce its uptake by mathematicians.

Keywords: Mizar, parser, compiler compilers, context free grammars

1 Language in mathematical knowledge management

Mathematical knowledge management (MKM) is concerned with making mathematics widely available to a multitude of users in a wide range of settings. In order to provide a comprehensive, internationally relevant repository of mathematics, it must be possible to translate between the many forms of language used in mathematics, including words, symbols and conventions. There is also the issue of how to convert existing mathematics into such an internationally robust form.

Formal mathematical languages seem to be generally recognised in the MKM community [2] as a good step towards providing a semantically grounded form of mathematics that can be used by machine-based MKM tools. Linguistically distinct forms of mathematics can then be generated through automatic translation from the formal language. To sustain growth of MKM repositories, mathematicians need to be able to contribute directly to such repositories. With a formal language basis, this means that mathematicians must be able to produce, or at least easily reproduce, their own mathematics as a formal language.

Our focus is on examining the gap between how mathematicians currently work and formal mathematical languages. Specifically, we examine one particular formal language, the Mizar language [11], and analyse aspects of its usability by mathematicians. Here 'usability' refers to those factors which affect the acceptance by the existing mathematical community, and the possible building up of a community of users from mainstream mathematics. There are other senses of ‘usability’ which we are not interested in here, e.g. reducing user error, except as they affect general acceptance of the language.

In the next section, we give particular details of why Mizar was chosen — largely because it is a substantial library with many good features. Following our analysis of issues and problems with the Mizar language, we discuss the more general implications for formal mathematical languages. Our key conclusions are that by being flexible and more ‘mathematical’ the language has become more difficult to understand, reducing the likelihood of wider acceptance by the mathematical community, and that further work is required to resolve these problems.

It must be stressed that whilst Mizar is used to highlight issues of formal languages, our intention is not to specifically critique Mizar but to draw general lessons from it. In fact, we regard Mizar as one of the more mathematically acceptable formal systems. This makes it particularly worth worrying about and improving still further.

2 Why Mizar?

Wiedijk recently collated examples of formal proofs developed on fifteen of the major formal systems [13]. His analysis of these proofs allowed a comparison between the different systems on a number of different factors. Naturally enough the systems vary and have a variety of strengths and weaknesses.

From this though, it was clear that Mizar has by far the largest library, the Mizar Mathematical Library (MML), which is available on-line as the Journal of Formalized Mathematics [8]. In addition it has a reasonably mathematical emphasis being a first order, classical logic with a terse language very easily read by mathematicians not familiar with the specifics of Mizar. For these reasons, Mizar stood out as a tool that mathematicians might be interested in using to write mathematics and therefore worth considering from a usability perspective.

Linguistically, the Mizar language also belongs to the class of mathematical vernaculars which has its ancestry in de Bruijn’s Automath system [10]. The various languages differ in particular features, however they are essentially formal, first order, mathematically oriented, weakly typed languages. Mizar is also a reasonable representative of this broader class of languages.

Mizar also has some practical features that make it appealing to study. There are several versions of Mizar for Windows and Unix platforms and the Windows version at least is quite straightforward to install and integrate with the emacs text editor. In addition there is the Mizar web-site [9] that provides lots of resources for understanding the Mizar language, the MML and the process of developing a Mizar article.

These considerations in some sense are secondary to the strengths of the Mizar system *per se*. However, in terms of usability and wide uptake of a system, any system that does not provide these sorts of services is automatically at a disadvantage.

This is not to say Mizar is without possible problems. Most notably, all advice on learning Mizar has mentioned the value of working closely with an experienced Mizar user. This suggests that there are idiosyncrasies in Mizar that are difficult avoid or master for all but the most dedicated solo worker.

3 Method of Analysis

In order to objectively analyse the Mizar language, a grammar was built from the description of the language on the Mizar web-site. The parsing environment used was JavaCC [7], a top-down compiler compiler. It uses a Java-like representation for lexical analysers and parsers [1] that it translates into native Java. JavaCC was chosen as being fairly representative of the various compiler compilers available.

Due to the difficulties of developing a parser for Mizar, initially only the abstracts were parsed though some articles have been parsed in their entirety[1].

Building a parser for a language is bread and butter to any computer scientist and does not usually constitute an analysis of the language. However, in this case, the Mizar language turned out to have some features that made standard parsing particularly difficult. Some of the features allow the language to function at a rich level, others can only be explained by an organic development of the system. Thus, parsing acted as a tool for 'systems forensics' giving insights into this language and formal languages more generally.

4 Parsing the Mizar language

The first thing to note about Mizar is that it is not a simple language. The grammar for Mizar has over 150 productions and 93 keywords. The Java language by contrast has only 83 productions and 50 keywords. This is not to say that Mizar is necessarily harder to learn or master because the difficulty of a language comes from its semantics rather than its syntax. It does, however, suggest that Mizar is certainly an interesting language to examine in detail.

From the work done, three key issues became clear and which considerably added to the complexity of building a parser. First and most surprisingly to us, Mizar is not a context free language. Secondly, equality and in particular iterative equality require a special semantics. Lastly, bracketing is every bit as complicated as the use of brackets in mathematics. We address each of these features in turn.

4.1 Context sensitivity

Context free grammars have been the basis and strength of formal languages since the 1960's, as exemplified in the development of Algol 60 [6]. Since then there has been a standardisation of tools and techniques so that developing new programming languages is

[1] JavaCC also has limitations which mean that, though in principle all 722 abstracts can be parsed, we only have parses for 491 of them (roughly 68%).

given as an exercise to first year undergraduates. Context free means that the language can be parsed using productions without reference to the meaning of individual tokens so long as they are correctly classified by the lexical analyser [1].

Mizar though is not context free. The parser requires three constructs in which the semantic meaning of tokens is required in order to correctly choose how to parse articles. The three constructs are:

1. Predicates in atomic formula expressions
2. Modes in type expressions
3. Structures in type expressions

In each of these cases, the construct is given by a token followed by a number of arguments and then a further construct. The problem occurs because both constructs are separated by commas as are the arguments of the first construct. It is impossible for the parser to decide whether the second construct really is a construct or simply an argument of the construct. For example, in `DECOMP_1` the following definition appears:

```
definition let X,Y be non empty TopSpace; let f be map of X,Y;
 attr f is s-continuous means...
```

Here `map` is a mode symbol taking two arguments X and Y. Without knowing this, Y could equally well be a further definition like f or an argument of map which it is. As argument lists can be arbitrarily long, the resolution of the possible ambiguity requires infinite lookahead which dramatically reduces parser performance.

The solution in our approach was to identify the number of arguments that could follow mode, structure and predicate constructs and to count that number of identifiers when parsing. When the total had been reached, it was clear that any subsequent identifiers must belong to the next construct. This is not fully reliable because overloaded predicates can take a variable number of identifiers. However, so far, it has been sufficient to allow *a* parse. The next step would be to do appropriate context based repairs.

4.2 Equality

Equality generally requires special treatment whenever it is used in logic [5]. This is partly because of its status as a special predicate that is able to re-label syntactic elements of the language and partly because of its huge overloading of meaning during the three centuries since Recorde defined the equals sign. Any mathematical language hoping to capture the varied and rich use of equals will inevitably have difficulties, and this is true of Mizar.

In Mizar terms, the = symbol is an in-fix predicate and indeed is defined as such in the special system article `HIDDEN`. Moreover, it is repeatedly redefined (overloaded) as such though usually with reference back to the system primitive version. However, in order to cope with the syntactic special place of equals, there are several productions that specifically treat = differently from other predicates. Though this complicates the language, it also allows the language to reflect some normal uses of =.

The use of = becomes complicated when required to do iterative equalities. This structure corresponds to producing proofs that look like:

$$\begin{aligned}(x+1)^2 &= (x+1).(x+1)\\ &= x.(x+1)+1.(x+1)\\ &= x^2+x+x+1\\ &= x^2+2x+1\end{aligned}$$

Here the left-hand side of later equalities is omitted being understood to be the same for all lines of the proof. Such structures are commonly used in mathematics and are reflected in other proof styles such as calculational proof [4].

Mizar implements iterated equality through the special symbol . = that would be used instead of the = symbol in all but the first line of the above calculation. This usage is natural to the reader and indeed the author of Mizar articles. From the parsing perspective though, it is somewhat tricky because a first line of proof that is an equality could either continue as an iterative equality or simply stand alone and the proof move on with a different structure. In the Mizar grammar given, iterative equality would require unlimited lookahead to look for the next possible . = symbol and hence know what sort of proof structure to parse. To avoid this, the only approach that seemed simple and effective was to allow a . = style proof line to follow *any* proof line. This allows the parser to proceed but of course introduces the possibility of having parsable but meaningless proofs. This of course would have to be repaired in a second parsing stage.

Certainly, this problem is not insurmountable and the current iterated equality provides a convenient and natural proof style for mathematicians. However, having had to re-think a part of the iterated equality grammar, it becomes clear that there are other sorts of iterated proof structures that mathematicians do use and that Mizar could usefully implement. For example, the iterated equivalents of $<$ and $\subseteq$ are commonly used. In Mizar terms, this could be dealt with via a special . prefix token that indicates an iterated predicate. It is not clear why Mizar does not have such a symbol — it probably reflects again the special nature of the = predicate in mathematics. The implementation of a general predicate iterators might best be done by uniform treatment of all predicates including equality.

4.3 Bracketing

Bracketing in Mizar is somewhat complicated. Naturally enough, Mizar has brackets to logically or presentationally separate parts of the grammar. These brackets include the usual suspects: {}, () and []. In addition, Mizar allows articles to define brackets that act as functors. For example, [] are used in the core Mizar syntax to indicate the arguments of certain predicates (privately defined ones) whereas they are also used to denote ordered pairs, triples etc.

This overloading does not cause parsing problems but instead lexing problems. Square brackets are defined in the lexer so that they can be referred to in the syntax for private predicates. However, they also need to be understood as possible functor brackets that have a completely different set of syntactic roles.

In effect, this overloading of brackets is just like normal mathematics. Some brackets are simply separators to help the reader interpret what belongs together. At other times, they mean something more and ambiguities ensue. For example, $f(a, b)$ in real analysis could equal be the function f acting on two arguments or the image of the real open interval from a to b under the function f. Mathematicians usually make efforts to clarify the meaning or ensure it can be inferred from the context.

The problem of brackets then is not truly serious but it does highlight the fact that Mizar is reflecting normal mathematical usage (with ensuing confusions) at the cost of complicating the machine semantics. Parsing through this overloading is not conceptually complicated but it does provide some careful implementation.

4.4 Some minor problems

There were also several minor problems that arose from using the Mizar web-site description of the Mizar language. In essence, the rules were nearly but not quite an accurate reflection of the language. Certain rules omitted possible tokens, such as the radix type can have an optional “non” to exclude certain types, so for example, from `TOPGRP_1`

```
 mode TopGroup is TopSpace-like Group-like associative (non empty TopGrStr);
```

The “non” is not included in the documentation’s description of radix type.

Or again, articles are required to have a DOS compatible name, in particular, to have at least five characters. This rule is clearly broken by the article `AMI1`.

More subtly, the `findvoc` utility that is able to find the definitions and uses of a particular term in the Mizar library is not entirely co-ordinated with the library. Some definitions found by `findvoc` do not in fact exist but are to be found elsewhere in the library. Clearly some refactoring has been done, but not across the entire set of Mizar services.

All of these problems are easily mended once they are recognised. The interesting point to note is that the description of the Mizar language is therefore manufactured anew rather than extracted directly from the Mizar system. Phrased another way, the manual is different from the system. This is a well-known source of problems for users, from aircraft control systems to video recorders [12]. In addition, it poses a large obstacle to any third party developers, like us, who wish to work on Mizar and develop their own tools.

It would be no surprise at all to learn that other systems have equally inaccurate descriptions and manuals.

5 Discussion

As stated out the outset, this critique is not intended to particularly criticise Mizar, but rather to learn from Mizar what the issues are for general formal languages. Mizar has some peculiar properties, but in fact the trade off with making a rich and flexible mathematical language is entirely appropriate. The whole area of MKM witnesses that mathematics is not the pure, unambiguous and invariant language that many non-mathematicians suppose it to be.

The three major parsing problems highlighted, namely context sensitive grammar, equality and bracketing, all allow Mizar to reflect real mathematical constructs. For example, mathematicians equate two topological spaces by equating simply the base set without reference to the actual topology. Mizar allows the same usage. Or similarly, mathematicians are often lazy about explicitly defining the number of arguments of a predicate or function. The context makes the exact meaning clear. The same is true in Mizar.

The underlying assumption seems to be that if a formal language is to be usable and acceptable to mathematicians then it must conform as closely as possible to accepted mathematical usage. Mizar has clearly adopted this approach and as such performs well in Wiedijk's study [13] as a significant language for formal mathematics. The drawback is that making the language usable has implications for its usefulness as an underlying representation of mathematics.

Banacarek has already pointed out that the useful and effective overloading of operators has implications for the formal search and retrieval tasks in the Mizar library [3]. Given that one of the obstacles to learning Mizar is mastering the extensive library, it is not desirable to make search and retrieval difficult. Search then must be context sensitive and depend on the perspective of the searcher not on some absolute meaning of a term or symbol. This poses challenges for being both usable by a human and usable by a search system.

Also, if a formal language is truly going to reflect current mathematical usage, it will need to be flexible not only today but in future. Mizar does allow the redefinition of terms so that they have the particular syntax required in a particular context. This could mean that a user is developing new mathematics using new terms or syntax for terms but is drawing on a resource that is comparatively old-fashioned. Whilst this is not a problem in the short term, in fact, is normal in mathematics, it may become a problem for the long term usability of the MML. For example, modern mathematicians have difficulty reading the intersection and union of sets denoted by . and + symbols, respectively. There is nothing wrong or ambiguous about this notation, it is simply out of date.

Our key point is that to be successful a system will need wide uptake. This can involve people developing specialised tools, services and adaptations for their particular needs. It would be impossible for a central Mizar team to provide all the services that a large user base could require. Instead, it would be better if third parties could develop their own software based on the Mizar core. Mizar clearly has a policy of making its system available in a variety of platforms. However, our problems with formulating a grammar for parsing, specifically the lack of an accurate description of the language, means that using and adapting the Mizar language is not the straightforward task it should be.

6 Conclusions

As it stands, the Mizar language is a flexible, mathematics-like language that can be used to specify formal proofs. Its flexibility makes it usable but also undermines its usefulness as a formal basis for mathematics. Several aspects of the language cannot be easily reconstructed, and this has obvious implications for potential users of the language, as well as those who wish to develop tools.

The Mizar language was designed for both people and computers. The difficulties we have highlighted raise the question of whether it is possible for a language to successfully fulfil both of these goals. We regard this an important open question for the MKM community.

It may be possible to adapt Mizar, or some other language, to give a more clearly defined and understandable language which suits both. On the other hand, it may be better if the languages of MKM divided into two: some languages would be ideal for mathematicians to use and develop mathematics, other languages would be ideal for storage, standardisation, search and retrieval tasks. Translation systems would ensure that what mathematicians produced would have formal correctness in the context of the larger formal repository. They would also translate sections of the repository back into the language context of the working mathematician.

Given our focus and inclination to usability, our future work is to adapt Mizar or a system founded on Mizar to become a language that mathematicians find useful and natural to use. In this sense, Mizar is pointing the way that we feel formal languages should go. Despite the problems we have discussed, and the fact that it is three decades old, it still represents the state-of-the-art in formal languages that could conceivably be adopted by the mainstream mathematical community.

Acknowledgements

Thanks to Prof. Harold Thimbleby for his useful comments. Jeremy Gow is funded on Prof. Thimbleby's Royal Society Wolfson Research Merit Award.

References

[1] A. V. Aho, R. Sethi & J. D. Ullman, *Compilers: Principles, Techniques and Tools*, Addison-Wesley, 1988

[2] A. Asperti, B. Buchberger & J. H. Davenport (eds), *Proc. of the 2nd International Conference on Mathematical Knowledge Management*. LNCS 2594, Springer, 2003

[3] G. Bancerek & P. Rudnicki, "Information Retrieval in MML," in [2], p119–132, 2003

[4] J. Grundy, "A Browsable Format for Proof Presentation," *Math. Universalis*, 2, 1996 `http://www.ant.pl/MathUniversalis/2/grundy/mu.html`

[5] A. G. Hamilton, *Logic for Mathematicians (revised edition)*, Cambridge, 1990

[6] E. T. Irons, "A syntax directed compiler for Algol 60," *ACM Compiler Symposium, 1960*, reprinted in *Comm. ACM*, 26(1):14–16, 1983

[7] Java Compiler Compiler, `http://javacc.dev.java.net/`

[8] Journal of Formalized Mathematics, `http://www.mizar.org/JFM/`

[9] Mizar Project Home Page, `http://www.mizar.org/`

[10] R. P. Nederpelt, J. H. Geuvers and R. C. de Vrijer (eds), *Selected Papers on Automath.* Studies in Logic and the Foundations of Mathematics, Volume 133, North-Holland, 1994.

[11] P. Rudnicki, "An overview of the Mizar project," in *Proceedings of 1992 Workshop on Types and Proofs for Programs*, 1992

[12] H. Thimbleby & P. B. Ladkin, "A Proper Explanation When You Need One," in *People and Computers X (HCI'95)*, pp107–118, Cambridge University Press, 1995

[13] F. Wiedijk, "Comparing Mathematical Provers," in [2], pp188–202, 2003