MECHANIZED MATHEMATICS AND ITS APPLICATIONS, VOL. 5, NO. 2, DECEMBER 2006, 1:10

New Auxiliary Software for MML Database Management

Robert Milewski

Institute of Computer Science
University of Bialystok
Sosnowa 64, Bialystok, Poland
milewski@math.uwb.edu.pl

Abstract – Mizar Mathematical Library (MML) [7] is a database containing more than 900 articles verified automatically by the MIZAR system making it one of the biggest databases of computer-checked mathematical knowledge in the world. The submission process of new articles, the great number of revisions and many experiments on the MML database depend on the existence of various auxiliary applications which have been created, usually ad hoc, since the beginning of the MIZAR system. With the start of some recent experiments with data-mining and research on the robustness of the MIZAR system [3], there is a need to create new software for realizing these tasks. This paper discusses the motivation for creating such programs and presents the design and reciprocal relationships between programs aimed at data-mining and research on the robustness of the MIZAR system.

1. Motivation and Main Ideas

MIZAR [1],[4],[8],[9] is a system for computer-aided formalization of mathematics created by Andrzej Trybulec in the 1970s. The development of the system ran in parallel to the development of a database of computer-checked mathematical knowledge. The first years of the MIZAR progress were dominated by numerous experiments which formed the main trends of the development. Since 1989 a database of computer-checked mathematical knowledge, Mizar Mathematical Library (MML), has been developed. MML continues to expand to this day and it now contains more than 900 articles verified automatically by the MIZAR system.

As with the Mizar system, MML is subject to permanent reorganizations, so called "revisions". Revisions are aimed at making it easier to use the information contained in MML and improving its order. Revisions have been made since the establishment of the MML database. This issue has resulted in the creation of a great number of auxiliary applications which were designed for such purposes as the rationalization of the revision process. Some auxiliary programs are so useful and popular that they are distributed with the MIZAR system (e.g., RELPREM, RELINFER).

In recent years, the idea of starting research on data-mining and evaluating the robustness of the MIZAR system has arisen [3]. This research makes it possible to create profiles of the MML database which contribute indirectly to the improvement of the MIZAR system and provide collections of data which may be used to compare the MIZAR system with other systems for computer-aided formalization of mathematics. Quite often, before starting such research, appropriate preparations of the MML database are necessary. For example, if there is a need to make experiments concerning references in MIZAR articles (altering their order or extending them with extra or redundant references), a problem arises with linking keywords "`then`"

Manuscript received September 8, 2005; revised November 16, 2006.

and "hence" because it is not possible to move such references to other positions. Solving this problem required implementing a program (called DELINKER) which replaces all occurrences of "then" and "hence" with references to the previous labelled sentence (in the case of linking with "hence", replacements had to be made with the word "thus", to preserve the conclusion) and which resolves all problems resulting from the changes. At the same time, we needed to implement a program (called LINKER) to reverse the work done by DELINKER.

This paper reports on auxiliary applications which have been developed in connection with the research on data-mining and the robustness of the MIZAR system.

2. Previously Available MIZAR Auxiliary Software

Since the beginning of the development of the MIZAR system, a few hundred applications designed to solve various problems have been created. However, some of them did not meet planned criteria while others became useless or obsolete and their maintenance was finally stopped in some system version. But many of them are still used intensively and maintained through many system versions. Some of them are distributed as extra utilities with the MIZAR system.

The list below presents the most useful MIZAR auxiliary applications:

- CONSTR - finds article names which should be placed in the CONSTRUCTORS directive in order to use given constructs from the MML database
- CHKLAB - points out unused labels in an article
- ERRFLAG - marks in a MIZAR-text-file the positions of errors reported by the verifier
- FINDVOC - finds names of vocabularies containing given entries
- INACC - finds unnecessary sentences in an article (which do not belong to any proof skeleton and have no references pointing to them)
- IRRTHS - detects unnecessary article names in THEOREMS and SCHEMES directives
- IRRVOC - detects unnecessary vocabulary names in VOCABULARY directive
- LISTVOC - displays all entries contained in a given vocabulary
- RELINFER - finds proof steps which may be replaced by one step (with all references from removed steps rewritten in the new inference step)
- RELITERS - finds fragments of iterative equalities which may be removed (after rewriting all references from the removed fragment to the next inference)
- RELPREM - indicates references which are not necessary to justify a given inference step
- REMFLAGS - removes the marks produced by ERRFLAG
- RENTHLAB - renames all label names in an article according to a fixed criterion
- SORT_REF - changes references in all inferences into alphabetical order
- TRIVDEMO - reports proof-end and now-end blocks which may by replaced by straightforward justification

Also worth mentioning is the whole set of applications designed for generating the *Journal of Formalized Mathematics* as well as for generating various MML statistics (e.g., the number of authors, the number of references to authors, the size of contribution by countries, the distribution of references, etc.).

3. References in the MIZAR System

Initial experiments concerning research on data-mining and the robustness of the MIZAR system concentrated around the concept of references. First, let us note that there are a couple of methods for justifying statements in MIZAR. In the simplest case, statements are accepted by the system automatically as a result of applying rules of propositional calculus, built-in information taken from requirements imported by a given article, or properties of definitions assigned to the used predicate or functor [5][6].

Another method of justifying the correctness of a sentence is by providing a formal proof of the given fact.

The third case, when a statement can be justified in one step by references to other facts, is called a straightforward justification. It means that after the statement we put the keyword "by" and then list the labels of sentences to which we want to refer which may include statements from the current article as well as external theorems. Additionally, we may refer to the previous statement by putting the word "then" before the justified statement:

```
A1: sentence1;
A2: sentence2;
sentence3;
then sentence4 by A1,A2,XBOOLE_0:def 1;
```

In the above example, we refer to `sentence1` and `sentence2` through labels `A1` and `A2`, respectively, and to `sentence3` through the use of "`then`" and the definition of the empty set `XBOOLE_0:def 1`. When we want to state the conclusion while referring to a previous statement we use the word "`hence`" (the combination "`then thus`" is not allowed).

There are no rules imposed on the order of references in a straightforward justification and it is left to the authors of the article.

4. "Hereby" as the Problematic Synonym for "thus now"

After some preliminary experiments, there was a need to label all sentences which can potentially be labeled in an article. It turned out that the word "`hereby`" (the synonym for "`thus now`") causes a problematic situation. The problem is that a sentence which begins with "`hereby`" may be linked by the use of "`then`" or "`hence`", but it cannot be labeled because the placement of a label would be between "`thus`" and "`now`". To manage this issue, it is necessary to replace all uses of "`hereby`" with "`thus now`" and simultaneously add a label between them.

In order to solve the problem, two rather uncomplicated applications were created: UNHEREBY and TOHEREBY. UNHEREBY changes all uses of "`hereby`" into "`thus now`", TOHEREBY inversely changes all uses of "`thus now`" (not separated by a label) into "`hereby`".

However, it is not true that both applications are mutually inverse because running them one after another on a MIZAR article does not return the original situation. This results from a lack of homogeneity in using "`hereby`" and "`thus now`". Some authors like using "`hereby`" while others prefer the expression "`thus now`". But there is a relationship between both applications, which is strong enough, namely they fulfill the Galois conditions:

```
u(t(u(A))) = u(A)
t(u(t(A))) = t(A)
```

where `t` means application TOHEREBY, `u` - UNHEREBY, and `A` - any MIZAR article. These applications are rather unsophisticated, but they both solve a significant problem on the way to other experiments.

5. Separation and Deseparation of Library References

Another problem encountered during experiments related to references concerning a feature of MIZAR grammar which allows authors to use shortened forms of library references. For example, an author may use the following shortened form in order to quote more than one external theorem from the same article:

```
by XBOOLE_1:2,3;
```

rather than:

```
by XBOOLE_1:2,XBOOLE_1:3;
```

In cases where the shortened form is used, it is impossible to, for example, freely change the order of references. Although it is possible to change the order of references into the shortened form, it is impossible to separate them with other references.

In order to solve this problem two applications were created: SEPREF and UNSEPREF. SEPREF changes all shortened forms of library references into full forms, while UNSEPREF inversely changes all full forms into shortened ones (only in the case where references from the same article occur next to each other to avoid altering their order).

Similarly as in the case described in the previous chapter, applications SEPREF and UNSEPREF are not mutually inverse, but they fulfill both Galois conditions:

```
u(s(u(A))) = u(A)
s(u(s(A))) = s(A)
```

where `s` means application SEPREF, `u` - UNSEPREF, and `A` - any MIZAR article. These applications are useful not only for experiments, but also for managing the form of articles in the MML database.

6. MIZAR Text Formatter

When we make any auxiliary applications we need to consider carefully whether or not they will significantly change the format of articles (e.g., addition of redundant spaces). Obviously it is possible to ignore this aspect and use a formatter which would format any article using a fixed set of formatting rules. But, there are at least two reasons why such a formatter application will not be used widely yet for a long time. The first of them is the fact that it is not possible to fix one universal style which would be observed by most of the users. It is possible

to impose that style, but it is a risky move. Articles may become less transparent for many users who do not prefer that fixed style. The second reason is the desire to preserve the original form of an article as contributed by the author.

Some auxiliary programs, however, cause some unnecessary editing changes, e.g., addition of redundant spaces, removal of multiple spaces (typed by the authors or added as a result of revisions) or gluing (splitting) of lines. It is the reason why the formatter was created. However, it is not an application used to standardize the MML database, but rather it is to be used in experiments (by comparing two formatted articles we avoid finding strictly editing differences). FORMATER edits MIZAR text according to a fixed set of formatting rules (which concern, for example, spaces between words, line breaks or indentation at the beginning of lines). It does not handle delimiting of line lengths. Lines have lengths resulting from the structure of sentences as well as from the rules which were fixed while FORMATER was created. In the traditional method of verifying a MIZAR article, the length of each line must not exceed 80 characters. But, it is possible to use an option which makes the verifier ignore the checking of line lengths. This option is also required when we want to continue working with articles which were modified by FORMATER. If, for some reason, it is necessary to fit an article to the 80 characters limit, a utility called LINE80 can be used to breaks lines so that their lengths are not longer than 80 characters. However, this is a program which should be treated as a work-application only (used only in experiments) because the method of searching for a place where a line break should occur is very simple. LINE80 simply breaks lines in the last possible position in a line without splitting whole words (otherwise it would cause syntactic errors). Still, it seems sufficient since the technical character of FORMATER does not require investing in a more sophisticated algorithm.

7. Linker and Delinker

To enable various modifications of references, it is necessary to eliminate links to previous statements which use the keyword "`then`" or those concluding with the word "`hence`". A reference introduced by "`then`" or "`hence`" is treated as a first reference in an inference step, but it cannot be simply moved to other positions. Solving the problem requires implementing a program to replace all occurrences of "`then`" with references to a labelled previous sentence and put it as a first reference in order not to change the original order. In the case of linking with "`hence`", it has to be replaced with the word "`thus`", to preserve the conclusion. Sometimes the previous statement needs an extra label when there is no label in the original text. To prevent conflicts, e.g., duplicating labels, the best solution seemed to be to change all label identifiers according to one naming scheme.

The DELINKER program inserts labels in every possible location and also changes the names of existing ones in the following way: the names of labels are of the form `Rx:`, where `x` is a consecutive number, starting from `1`. DELINKER also removes linking with "`then`" and changes "`hence`" into "`thus`", adding at the beginning a reference to the last accessible label.

To enable the labelling of all possible sentences, we ran the UNHEREBY application before all of the operations listed above to change all occurrences of "hereby" into "thus now".

The same fragment of [2] before and after using DELINKER is presented in the following example.

Before running DELINKER:

```
Lm3:
 for x,A,B,C,D being set holds
  x in A\/B\/C\/D iff x in A or x in B or x in C or x in D
 proof
  let x,A,B,C,D be set;
  hereby assume x in A\/B\/C\/D;
   then x in A\/B\/C or x in D by XBOOLE_0:def 2;
   then x in A\/B or x in C or x in D by XBOOLE_0:def 2;
   hence x in A or x in B or x in C or x in D
      by XBOOLE_0:def 2;
  end;
  assume x in A or x in B or x in C or x in D;
  then x in A\/B or x in C or x in D by XBOOLE_0:def 2;
  then x in A\/B\/C or x in D by XBOOLE_0:def 2;
  hence thesis by XBOOLE_0:def 2;
 end;
```

After running DELINKER:

```
R5:
 for x,A,B,C,D being set holds
  x in A\/B\/C\/D iff x in A or x in B or x in C or x in D
 proof
  let x,A,B,C,D be set;
  thus R6: now assume R7: x in A\/B\/C\/D;
   R8: x in A\/B\/C or x in D by R7,XBOOLE_0:def 2;
   R9: x in A\/B or x in C or x in D by R8,XBOOLE_0:def 2;
   thus R10: x in A or x in B or x in C or x in D
      by R9,XBOOLE_0:def 2;
  end;
  assume R11: x in A or x in B or x in C or x in D;
  R12: x in A\/B or x in C or x in D
     by R11,XBOOLE_0:def 2;
  R13: x in A\/B\/C or x in D by R12,XBOOLE_0:def 2;
  thus R14: thesis by R13,XBOOLE_0:def 2;
 end;
```

To be able to restore changes produced by DELINKER, we needed to create a "reverse" application (called LINKER). The specifications of LINKER are:

- to change label references to directly preceding statements into links using the keyword "then" or "hence"
- to remove unused labels and change the format of used labels into one just like after running DELINKER
- to change all occurrences of "thus now" (not separated by a label) into "hereby"

Because some of these tasks are already done by other existing utilities, creating a program realizing all of these tasks seemed unnecessary. There is for example an application called CHKLAB which removes unused labels as well as RENTHLAB which gives all labels in a uniform style according to the following principle (increasing indentations represent "deeper" proof levels):

```
Th1: .........;
   A1: .........;
   A2: .........;
      A3: .........;
   A4: .........;

Th2: .........;
   A1: .........;
      A2: .........;
   A3: .........;
```

In order to use the applications mentioned above to do the linking, we can imagine the following calling scheme:

- LINKER - an application which finds references by a label to directly preceding statements and changes them into links using the keywords "then" or "hence"
- CHKLAB
- RENTHLAB
- TOHEREBY

The LINKER and DELINKER applications are not mutually inverse, as in the case of the pairs of programs described earlier. Therefore, it may be interesting to investigate what connections exist between them (e.g., do they meet the Galois conditions?). Here writing LINKER means running the series of programs as described above: LINKER - CHKLAB - RENTHLAB - TOHEREBY.

Checking the first Galois condition (`d` - DELINKER, `l` - LINKER, `A` - any MIZAR article):

```
d(l(d(A))) = d(A)
```

we obtain the following difference in size (in bytes) of the MML database after processing:

d(A)	66 220 737
d(l(d(A)))	66 556 963
difference	336 226

As we can see, there is a significant difference in size (the application of `d*l*d` causes the text to be lengthened as compared to the text which was processed only through DELINKER). There are big differences in contents as well and they follow not only from editing problems (addition of spaces and changing multiple lines into single ones), but also from the fact that using LINKER on the unlinked databases may unintentionally alter the order of references (though this issue does not influence the size of the database, only the differences in contents). If there is a reference to a directly preceding statement, but it is not at the first position on the list of references, then changing it into a link using the keyword "`then`" or "`hence`" (which always appears first on the reference lists) causes a permutation of the list as shown below.

Before running LINKER:

```
R1: x in A;
R2: A c= B;
R3: x in B by R1,R2;
```

After running LINKER:

```
R1: x in A;
R2: A c= B;
then R3: x in B by R1;
```

After running DELINKER:

```
R1: x in A;
R2: A c= B;
R3: x in B by R2,R1;
```

Checking the second condition:

```
l(d(l(A))) = l(A)
```

we obtain the following results:

l(A)	61 606 318
l(d(l(A)))	62 052 165
difference	445 847

The increase in size is even greater than before and there are greater differences in contents as well. This results only from editing problems (addition of spaces and changing multiple lines into single ones). There is no problem of changing reference orders. It follows from the fact that both the left and right sides of an equality are processed at least one time through LINKER.

The order of references may be different with relation to the original text, but it is the same when using only LINKER and when using applications LINKER - DELINKER - LINKER.

The result of this experiment suggests that adding the FORMATER application to both sides of the Galois conditions as shown below will make a full equality:

```
f(d(l(d(A)))) = f(d(A))
f(l(d(l(A)))) = f(l(A))
```

The experiment yields the following results:

f(d(A))	60 005 690
f(d(l(d(A))))	60 005 690
difference	0
f(l(A))	55 396 566
f(l(d(l(A))))	55 396 566
difference	0

In both cases the size of the MML database is identical. In the second condition, no differences in contents result, therefore we can state that the Galois condition corrected with FORMATER is really true. But in the first condition, in spite of the equality of sizes, there are differences in contents. FORMATER eliminated the editing problems, but the problem of altering the order of references by LINKER remains. In this situation it is necessary to enrich both sides of the condition with a call of LINKER. It may be added at the beginning or end of both sides of our condition:

f(l(d(A)))	55 396 566
f(l(d(l(d(A)))))	55 396 566
difference	0
f(d(l(A)))	60 005 690
f(d(l(d(l(A)))))	60 005 690
difference	0

In both cases there are no differences in size as well as in contents.

Another conclusion which comes to mind after the execution of the experiments is that it seems impossible to reverse exactly the work done by DELINKER (and obviously to reverse LINKER). This is because of the less important editing problems, but first of all because of the fact that the MML database is not formatted (not only with respect to editing, but also with respect to the contents) according to concrete, fixed, and unambiguous rules. The format of MML depends on the preferences of individual authors. It concerns the problems of linking (it is not obligatory to use the keywords "`then`" or "`hence`" when adding a reference to the directly preceding statement), order of references, use of equivalent expressions, as well as many other situations not described above.

8. Final Remarks

The issues connected with references in the MIZAR system are only the beginning of experiments which concern research on the robustness and data-mining of the MIZAR system. Future experiments will require creating additional auxiliary applications. It is very important to check before creating new applications whether or not the functions can be found in the existing set of auxiliary programs in a different form. Some applications may be the same as that which is wanted or they may fulfill at least a part of a task needed. The care of application functionality and not repeating programs is as important as creating new software.

The applications created by the author in this work (in particular DELINKER and LINKER) will be widely used by other users of the MIZAR system and they will also become useful tools for carrying out various experiments as well as revisions of the MML database.

Acknowledgments

I thank all of those who helped me with writing this paper for their valuable hints and advice. In particular I want to thank A. Trybulec, A. Naumowicz, and G. Bancerek.

References

[1] E. Bonarska, *An Introduction to PC Mizar*, Fondation Ph. le Hodey, Brussels, 1990.
[2] A. Korniłowicz and R. Milewski, *Gauges and Cages. Part II*, Formalized Mathematics, Vol. 9, No. 3, 2001, pp.555–558.
[3] R. Milewski, *Robustness of Systems for Formalizing Mathematics - Testing Monotonicity and Permutability of References in* MIZAR, Mechanized Mathematics and Its Applications, Vol. 4, No. 1, 2005, pp.51–58.
[4] M. Muzalewski, *An Outline of PC Mizar*, Fondation Philippe le Hodey, Brussels, 1993.
[5] A. Naumowicz and C. Byliński, *Basic Elements of Computer Algebra in* MIZAR, Mechanized Mathematics and Its Applications, Vol. 2, No. 1, 2002, pp.9–16.
[6] A. Naumowicz and C. Byliński, *Improving* MIZAR *Texts with Properties and Requirements*, MKM 2004, Springer-Verlag Berlin Heidelberg, 2004, pp.290–301.
[7] P. Rudnicki, *An Overview of the Mizar Project*, Proceedings of the 1992 Workshop on Types for Proofs and Programs, Chalmers University of Technology, Bastad, 1992.
[8] P. Rudnicki and A. Trybulec, *On Equivalents of Well-foundedness. An experiment in Mizar*, Journal of Automated Reasoning, 23, Kluwer Academic Publishers, 1999, pp.197–234.
[9] A. Trybulec, *Some Features of the Mizar Language*, ESPRIT Workshop, Torino, 1993.