Evaluating Prospective Built-in Elements of Computer Algebra in MIZAR*

Adam Naumowicz

Institute of Computer Science
University of Białystok, Poland
adamn@mizar.org

Abstract. In this paper we evaluate the benefit of using MIZAR built-in routines devised to automatically deduce some inferences, called `requirements` in the MIZAR terminology. This data is compared with the potential impact of implementing several new `requirements` - almost entirely based on internal procedures already present in the MIZAR source code - namely: `Div`, `Mod`, `Divides`, `GCD`, `LCM`, `Power` and `Factorial`. Some technical aspects of the `requirements` technology are also presented in detail.

1 Introduction

The philosophy behind the MIZAR system has always been to develop a system useful for performing various aspects of mathematical practice in a computer environment (see [2]). This gave rise to a formal language readable for both humans and computers and a system for verifying correctness of reasoning steps. Although the system has been evolving for over 30 years now, there is a lot of space for improvement, especially in the realm of available automation of basic calculations, directed at approaching the very much desirable Poincaré principle (see [10]). The still-too-high value of the so-called De Bruijn factor ([8]) measured for more advanced formal encodings ([3]) indicates the need of further strengthening the power of "obviousness" within MIZAR's CHECKER. In this paper we present ongoing research in this direction by means of implementing internal MIZAR routines which help verify reasoning steps based on frequently used basic concepts - numerical computations, boolean operations on sets, etc.

We assume that the reader is familiar with the theoretical fundamentals of MIZAR's CHECKER as presented in [9] and recognizes the functions of its main modules: PRECHECKER, EQUALIZER and UNIFIER.

In Table 1 there are collected the numbers of calls to each of these modules issued during a complete VERIFIER run on the whole Mizar Mathematical Library (MML). These numbers show the size of the database, and on the other hand, should form a point of reference for the CHECKER's add-ons described in the sequel.

* This work has been partially supported by the FP6 IST project no. 510996 *Types for Proofs and Programs* (TYPES).

ISBN 978-83-7431-128-1 ISSN 0860–150X

Table 1. CHECKER statistics for entire MML (MIZAR ver. 7.8.03, MML ver. 4.76.959)

Module	Number of calls
PRECHECKER	763420
EQUALIZER	958525
UNIFIER	1101031

A few words of explanation are due here: the disjunctive normal form of formulae occurring in the justified inference, which is produced in PRECHECKER, usually consists of one or two disjuncts, so naturally the number of EQUALIZER's calls is greater than that of PRECHECKER's. However, one may ponder why the number of UNIFIER's calls is greater than EQUALIZER's, as there are inferences verified solely by EQUALIZER without running UNIFIER at all. Indeed, out of the total 958525 inferences passed into EQUALIZER, 154902 are verified there in which case UNIFIER is not invoked. But one must know that CHECKER may try to unify several universally quantified formulae, one by one, if more than one such formulae appear in an inference step and verification of previous ones fails. Another point here is that UNIFIER is also called to instantiate variables in formulae within Fraenkel terms in much the same way.

2 Requirements

The most effective technique of strengthening CHECKER's power has proved to be the identification of selected frequently used notions and implementing special internal routines which recognize the notions' constructors and offer extra processing according to their specific properties. Some form of special processing of numerical data was introduced in MIZAR as early as the beginning of 1980s in the implementation of MIZAR-2 ([2]).

More recently, many improvements have been added to automate as much as possible simple reasoning steps concerning operations on numbers and sets. There are now five sets of such internally-identifiable constructors - `BOOLE`, `SUBSET`, `NUMERALS`, `ARITHM` and `REAL` - MIZAR users can include these names in the environment directive `requirements` of their articles to benefit from their special processing. For a comprehensive account of these `requirements` directives one should refer to [4] or [5]. The most recent improvement, which these papers do not cover, is the new realization of the `requirements ARITHM` directive thoroughly reimplemented by C. Byliński. In this new implementation all terms equipped with the attribute `complex` are subject to decomposition with regard to basic arithmetic operations - it results in all equality classes in EQUALIZER having a polynomial representation where monomials are formed by "irreducible" `complex` terms - linear algebra methods are then used to find potential equalities between different classes.

Although the `requirements` are not a "regular" language construct, so users cannot create their own ones without reimplementing the MIZAR software, it is highly desirable that users knew more about this technique. Understanding how

`requirements` work is important for all those who are interested in using MIZAR for various special purposes - to name just one such case, using MIZAR for educational purposes may require the ability to "turn on and off" selected requirements or to build a small local database from scratch, but be able to use MML `requirements` concerning numbers (such an example of MIZAR usage is presented in [6]). Table 2 shows current built-in `requirements` and their internal numbers (implemented as an enumerated type - if one of the internal numbers is assigned to an appropriate constructor, then the special processing associated with that number is performed for all instances of that constructor).

2.1 The structure of `requirements` files

The `requirements` are imported by users to their articles in the same manner as all other library items. The information is stored in the database as `*.dre` files. Thanks to the work of Josef Urban, all MIZAR library files are currently available in an XML form (see [7]). Their content is easy to parse and grammar is well documented - one can find the documentation of `requirements` files in the 'doc' folder within the MIZAR distribution, e.g. on Unix platforms the file `/usr/local/doc/mizar/xml/Mizar.html#Requirements`. Requirements may be consulted to see that the content of `requirements` files is the following:

```
%Signature;, ( %Requirement; )*
```

where `Signature` is a list of articles from which constructors should be imported and `Requirement` is a description of a constructor specially treated by the system. The description contains its internal number and optionally its name, the article's id (`aid`) and position in the article (`absnr`) - see Table 3 for a complete description.

2.2 Example - `requirements HIDDEN`

Here is the contents of the file `hidden.dre` which is automatically imported by the ACCOMMODATOR to properly associate the most basic MIZAR notions:

```
<?xml version="1.0"?>
<Requirements>
  <Signature>
    <ArticleID name="HIDDEN"/>
  </Signature>
  <Requirement constrkind="M" constrnr="1" nr="1"/>
  <Requirement constrkind="R" constrnr="1" nr="2"/>
  <Requirement constrkind="R" constrnr="2" nr="3"/>
</Requirements>
```

Table 2. Currently implemented `requirements` (MIZAR ver. 7.8.03, MML ver. 4.76.959)

No.	Requirements file	MML definition	Short description
0	-	-	internal variable - not associated with any constructor
1	`hidden.dre`	`HIDDEN(:def 1)`	automatically set to the first mode constructor[a]
2	`hidden.dre`	`HIDDEN(:def 2)`	the absolute equality relation
3	`hidden.dre`	`HIDDEN(:def 3)`	the belonging relation
4	`boole.dre`	`XBOOLE_0:def 5`	the attribute `empty`
5	`boole.dre`	`XBOOLE_0:def 1`	the empty set
6	`subset.dre`	`SUBSET_1:def 2`	element of a set
7	`subset.dre`	`ZFMISC_1:def 1`	the power set
8	`subset.dre`	`TARSKI:def 3`	set-theoretic inclusion
9	`subset.dre`	`SUBSET_1(redef.)`	element of a subset of a set
10	-	-	the set of real numbers[b]
11	-	-	the set of natural numbers[c]
12	`arithm.dre`	`XCMPLX_0:def 4`	the addition of complex numbers
13	`arithm.dre`	`XCMPLX_0:def 5`	the multiplication of complex numbers
14	`real.dre`	`XXREAL_0:def 5`	the <= relation on real numbers
15	`numerals.dre`	`ORDINAL_1:def 1`	the successor of a natural number
16	`boole.dre`	`XBOOLE_0:def 2`	the union of two sets
17	`boole.dre`	`XBOOLE_0:def 3`	the intersection of two sets
18	`boole.dre`	`XBOOLE_0:def 4`	the difference of two sets
19	`boole.dre`	`XBOOLE_0:def 6`	the symmetric difference of two sets
20	`boole.dre`	`XBOOLE_0:def 7`	true if two sets have an empty intersection[d]
21	`arithm.dre`	`XCMPLX_0:def 6`	the negated complex number
22	`arithm.dre`	`XCMPLX_0:def 7`	the inverse complex number
23	`arithm.dre`	`XCMPLX_0:def 8`	the difference of complex numbers
24	`arithm.dre`	`XCMPLX_0:def 9`	the division of complex numbers
25	-	-	the attribute `real`[e]
26	`real.dre`	`XXREAL_0:def 6`	the attribute `positive`
27	`real.dre`	`XXREAL_0:def 7`	the attribute `negative`
28	-	-	the attribute `natural`[f]
29	`arithm.dre`	`XCMPLX_0:def 1`	the imaginary unit
30	`arithm.dre`	`XCMPLX_0:def 2`	the attribute `complex`
31	`numerals.dre`	`ORDINAL_1:def 12`	the set of natural numbers

[a] `set` by default.
[b] Implementation discontinued.
[c] Implementation discontinued, see No. 31.
[d] Implementation postponed.
[e] Implementation discontinued.
[f] Implementation discontinued.

Table 3. The grammar of `*.dre` files (MIZAR ver. 7.8.03)

Attribute	Data type	Use
constrkind	Enumeration: "M" | "L" | "V" | "R" | "K" | "U" | "G"	Obligatory
constrnr	xsd:integer	Obligatory
nr	xsd:integer	Obligatory
reqname	xsd:string	Optional
absnr	xsd:integer	Optional
aid	xsd:string	Optional

2.3 Assessment of `requirements` usage

Several methods could be used to assess the usefulness of specific `requirements` for the authors of new articles and the development of MML. First, the available debug information (from CHECKER's reports) could be used since all relevant occurrences of the `requirements` enumeration type are marked in the MIZAR source code. This, however, may not be enough to state if a given action really resulted in accepting otherwise unacceptable inferences, e.g. joining two equality classes may happen in some "later" part of the EQUALIZER code.

With the access to the MIZAR source code, one could also recompile the ACCOMMODATOR program (`accom`) to disable specific `requirements` by preventing their numbers to be printed into the `*.ere` file which is later used by the VERIFIER. Or similarly, one could post-edit the `*.ere` file produced by the ACCOMMODATOR, which is a sequence of numbers representing constructor numbers associated with subsequent `requirements` numbers - both this solutions are not very "clean".

Much more "elegant" approach seems to be the use of standard MIZAR technique of building local environment - knowing the structure of `requirements` files, one may prepare a local version of a relevant *.dre file with specific lines removed that correspond to a given constructor. This method is easily reproducible by any user and requires no special tools. This way all MML `requirements` can be tested in two steps:

1. one `requirement` disabled each time,
2. full library check with standard `verifier`.

The results of such a test are collected in Table 4. Here is some explanation for the notions measured in this test:

– Possible applications (number of articles that follow the article which introduces `requirements` constructors according to the `mml.lar`[1] ordering, excluding the "documentation" articles[2]

[1] More information to be found in the archives of the MIZAR-Forum mailing list, e.g. http://mizar.uwb.edu.pl/forum/archive/0609/msg00076.html

[2] These are articles accompanying every requirements file, e.g. `boole.miz`, that contain proves of built-in facts - kept in MML only to help preserve the system's consistency.

- Dependent articles (number of articles that indeed rely on specific `requirements`, i.e. there were errors reported in the test)
- Total number of errors
- Average number of errors (calculated for all articles that depend on specific `requirements`)

Table 4. `requirements` usage statistics (MIZAR ver. 7.8.03, MML ver. 4.76.959)

No.	Possible applications	Dependent articles	Total no. of errors	Avg. no. of errors
1	958	958	0[a]	0
2	958	952	343176	360.48
3	958	895	92322	103.15
4	956	879	43080	49.01
5	956	539	3994	7.41
6	951	913	372186	407.65
7	951	857	50745	59.22
8	951	773	20025	25.90
9	951	663	299850	452.26
12	920	564	34051	60.37
13	920	382	14081	36.86
14	917	512	19735	38.54
15	933	20	45	2.25
16	956	169	858	5.07
17	956	112	392	3.50
18	956	86	186	2.16
19	956	1	5	5.00
21	920	315	4770	15.14
22	920	30	187	6.23
23	920	463	16350	35.31
24	920	191	3215	16.83
26	917	377	8463	22.45
27	917	367	7613	20.74
29	920	11	64	5.82
30	920	545	19681	36.11
31	933	668	300488	449.83

[a] In the current implementation, this `requirement` is preset in the code and cannot be turned off without recompiling the ACCOMMODATOR - the MIZAR type system depends on the assumption that the type created with the `mode` constructor number 1 is the "widest" type.

3 New `requirements` proposals

Obviously, the currently implemented set of `requirements` by no means covers the needs of Mizar users. There are still many operations which users would like to have automatically calculated for them by the system. Below we present a selection of potentially desirable built-in constructors together with assessment of their usefulness based on a rudimentary implementation. Worth noticing is the fact that most of the code required by the implementation has already been present in Mizar sources as part of `requirements ARITHM` supporting the arithmetic of arbitrarily long integers. Namely, we present the implementation of the following numerical functions: `Div`, `Mod`, `Divides`, `GCD`, `LCM`, `Power` and `Factorial`.

3.1 Usage assessment for new `requirements`

Checking which MML inferences are accepted thanks to new `requirements` requires the whole database to be "clean" with respect to the following utilities reporting unnecessary items in reasoning:

- `relprem` (reports unnecessary premises)
- `relinfer` (reports unnecessary inference steps)
- `reliters` (reports unnecessary inference steps within iterative equalities)

The Mizar toolbox contains utilities that allow to automatically eliminate all errors reported by `relprem` and `reliters`. At the moment, there is no working program available to do the same with `relinfer`'s errors - in this case the test results have been "simulated" by subtracting the number of errors reported before using the new `requirements` from and number obtained while using them. All tests have been carried out on an Intel(R) Pentium(R) 4 (3.00GHz, 2GB RAM) workstation running 2.6.16.21-0.25-smp i386 GNU/Linux. In such an environment verification of the whole MML with `verifier` currently takes about two hours, while a complete run of `relprem`, `relinfer` and `relites` takes about four, three, and one and a half hours, respectively.

3.2 `requirements NAT_D`

The article `NAT_D` appeared in the MML as a result of a revision aimed at better organization of the various divisibility-oriented notions - it contains the redefinitions for natural numbers, whereas the original constructors are introduced in the article `INT_1`. Table 5 summarizes the constructors used.

The implementation of `Div`, `Mod`, `GCD`, and `LCM` is rather straightforward - the value is calculated by an appropriate function for all instances of terms with a given constructor and arguments having a numerical value in the Equalizer - as a result some equality classes may become equal or a contradiction may be found. The `Divides` code is slightly more complicated - in the Equalizer we must check all instances of positively and negatively valuated formulae built with the `Divides` constructor and calculate if they agree with the numerical values of

Table 5. requirements NAT_D (MML ver. 4.76.959)

No.	Requirements file	MML definition	Short description
32	nat_d.dre	INT_1:def 7	the integer division
33	nat_d.dre	INT_1:def 8	the remainder of integer division
34	nat_d.dre	NAT_D:def 4	the least common multiple of two natural numbers
35	nat_d.dre	NAT_D:def 5	the greatest common divisor of two natural numbers
36	nat_d.dre	INT_1:def 9	true if and only if one integer divides another

their arguments, but also provide some code in the `Unifier` which tries to pattern-match free variables in appropriate formulae with a contradictory substitution (as described in [9]).
Conforming to the grammar of `*.dre` files presented in Section 2.1 we encoded the file `nat_d.dre` as follows:

```
<?xml version="1.0"?>
<Requirements>
  <Signature>
    <ArticleID name="HIDDEN"/>
    <ArticleID name="INT_1"/>
    <ArticleID name="NAT_D"/>
  </Signature>
  <Requirement constrkind="K" constrnr="5" nr="32"/>
  <Requirement constrkind="K" constrnr="6" nr="33"/>
  <Requirement constrkind="K" constrnr="11" nr="34"/>
  <Requirement constrkind="K" constrnr="13" nr="35"/>
  <Requirement constrkind="R" constrnr="4" nr="36"/>
</Requirements>
```

Below are the results of testing the usage of each of these `requirements`:

Table 6. Number of "irrelevant" items reported by library utilities (MML ver. 4.76.959)

No.	relprem	relinfer	reliters	Total:
32	59 (16 files)	19 (7 files)	8 (5 files)	43
33	224 (15 files)	76 (6 files)	40 (5 files)	340
34	4 (1 file)	0	0	4
35	54 (4 files)	0	45 (1 file)	99
36	201 (9 files)	115 (8 files)	0	316

3.3 `requirements NEWTON`

The current MIZAR code does not contain functions computing the value of `Power` and `Factorial`. However, their implementation is trivial in presence of multiplication for arbitrarily long integers, so can easily be added. Then, the CHECKER implementation of these `requirements` resembles that of e.g. `Div` or `Mod`. The associated constructors are listed in the table below.

Table 7. `requirements NEWTON` (MML ver. 4.76.959)

No.	**Requirements file**	MML **definition**	**Short description**
37	`newton.dre`	`NEWTON:def 1`	the exponentiation (complex base, natural exponent)
38	`newton.dre`	`NEWTON:def 2`	the factorial of a natural number

A corresponding `newton.dre` file should have the following form:

```
<?xml version="1.0"?>
<Requirements>
  <Signature>
    <ArticleID name="HIDDEN"/>
    <ArticleID name="NEWTON"/>
  </Signature>
  <Requirement constrkind="K" constrnr="2" nr="37"/>
  <Requirement constrkind="K" constrnr="5" nr="38"/>
</Requirements>
```

Finally, Table 8 presents the results of checking the usefulness of implementing special processing for `Power` and `Factorial`:

Table 8. Number of "irrelevant" items reported by library utilities (MML ver. 4.76.959)

No.	`relprem`	`relinfer`	`reliters`	**Total:**
37	319 (33 files)	68 (19 files)	239 (26 files)	626
38	102 (14 files)	0	38 (8 files)	140

This implementation caused some performance problems - apparently there are inferences in the MML in which quite large exponents are used (e.g. 3^{32768}, i.e. 15635 digits!) - normally the inference gets justified with a universal premise, but with a `requirement` turned on for the `Power` function, the verification takes quite a lot of time. The same may as well concern the `Factorial` implementation.

4 Conclusions

After performing the tests of usability for the prospective `requirements`, it seems that they could be a valuable add-on for the Mizar Verifier in the process of approaching the Poincaré principle. Some of them, however, must be introduced carefully, as there may occur performance issues - in these cases the principle should probably be treated as "a dynamic principle which puts it in the hands of a human user whether to expand an argument in different ways" in the spirit proposed in [1]. The application of optimized libraries or external programs could be another solution to these problems.

References

1. Kerber, M., A Dynamic Poincaré Principle, In J. M. Borwein and W. M. Farmer (Eds.), MKM 2006, LNAI 4108, pp. 44–53, 2006.
2. Matuszewski, R. and P. Rudnicki, Mizar: the First 30 Years, Mechanized Mathematics and Its Applications, Vol. 4 (1), pp. 3–24, 2005, http://mizar.org/people/romat/MatRud2005.pdf.
3. Naumowicz, A., An Example of Formalizing Recent Mathematical Results in MIZAR. In C. Benzmüller (Ed.), Towards Computer Aided Mathematics, Journal of Applied Logic, 4(4), pp. 396–413, Elsevier, 2006.
4. Naumowicz, A. and C. Byliński, Basic Elements of Computer Algebra in MIZAR, Mechanized Mathematics and Its Applications, Vol. 2, pp. 9–16, 2002.
5. Naumowicz, A. and C. Byliński, Improving Mizar Texts with Properties and Requirements, In A. Asperti et al. (Eds.), MKM 2004, LNCS 3119, pp. 290–301, 2004.
6. Retel, K. and A. Zalewska, Mizar as a Tool for Teaching Mathematics, Mechanized Mathematics and Its Applications, Vol. 4(1), pp. 35–42, 2005.
7. Urban, J., XML-izing Mizar: Making Semantic Processing and Presentation of MML Easy, In M. Kohlhase (Ed.), MKM 2005, LNAI 3863, pp. 346–360, 2006.
8. Wiedijk, F., The De Bruijn Factor, http://www.cs.ru.nl/~freek/factor/factor.pdf .
9. Wiedijk, F., Checker, http://www.cs.ru.nl/~freek/mizar/by.pdf .
10. Wiedijk, F. (ed.), The Seventeen Provers of the World (foreword by Dana S. Scott), LNAI 3600, 2006.