Towards Standardized Mizar Environments

Adam Naumowicz

Institute of Informatics, University of Białystok,
Konstantego Ciołkowskiego 1M, 15-245 Białystok, Poland
adamn@math.uwb.edu.pl

Abstract. The effectiveness of formalizing substantial parts of mathematics largely depends on the availability of relevant background knowledge. The bigger the knowledge library, however, the harder it is to specify what is or should be relevant. Even with today's size of the libraries available for various proof assistants, importing the whole library is not an option for practical performance reasons. On the other hand, too detailed import machanisms are prone to dependency problems and pose certain difficulty for the user. In this paper we present the key ideas of a project aimed at generating standardized formalization environments which could be used to facilitate developing new formalizations based on the current content of the Mizar Mathematical Library.

1 Introduction

Accumulating and reusing all previously formalized data in a form of a standard library is a necessity for today's proof assistant systems involved in large formalization projects. The way in which a given system allows to import available data from its library varies across most proof assistants with different foundations and implementation. But a common feature seems to be a file-based library items organization. For example, HOL Light has the import handled by its metalanguage OCaml's `#use` and the `needs` directive [1], there is the `Require [Import|Export]` in Coq [2], or the `imports` directive in Isabelle/Isar [3] and so on. In the current Mizar [4, 5] system[1] there are several importing directives with different roles and semantics. Especially for users of other systems, the original Mizar importing method based on a set of rather fine-grained import directives might seem cumbersome and unnecessarily complicated. This is the main motivation for the extension of the standard Mizar utilities and a corresponding language addition presented in this paper. However, the proposed solution is not meant to completely replace the current importing methods, but rather help the users to easily start writing Mizar formalizations without too much problems with setting-up a working environment containing required items from the standard library.

It should be noted that the original Mizar importing mechanism comes from the times of rather scarce computer resources (RAM, storage, computational

[1] `http://mizar.org`

power), when it was designed as a practical solution to cope with these limitations. However, despite its intricacy, it still works reasonably well nowadays when the complexity of the formalized developments grows. Maintaining the collection of interdependent articles forming the Mizar Mathematical Library (MML) based on these fine-grained dependencies is one of the factors that most of the current complex formalization developments are still processed in seconds or minutes rather than hours, which is sometimes the case with other proof environments. Note also, that even more fine-grained access to every particular definitions and proofs can sometimes be desirable, as shown by the `mizar-items` project [6] aimed at reconstructing the prerequisites of theorems in the spirit of reverse mathematics.

2 Mizar Library Importing Specifics

Let us briefly recall here that an environment of a typical Mizar article contains a series of directives importing various sorts of notions from articles previously available in the MML (or a local data base). Basic information about each directive can be found in the Mizar user's manual [7].

The directives are `vocabularies`, `notations`, `constructors`, `registrations`, `requirements`, `definitions`, `equalities`, `expansions`, `theorems` and `schemes`. Apart from `requirements` and `vocabularies`, each directive contains a list of (usually a couple dozen) article names whose data should be imported to give a certain meaning to the newly developed formalization. For a detailed explanation of the functional and implementational aspects of the more advanced directives: `registrations`, `reductions`, `equalities` and `expansions`, the reader may consult [12], [13] , [14] and [15], respectively.

The `requirements` directive accepts as its arguments specific names which do not directly correspond to any formalization article, but instead provide means to switch on special processing for selected highly-used notions, e.g. `BOOLE` for automating Boolean operations on sets [8], or `ARITHM` for automating the arithmetic of real and complex numbers [9].

Originally, the `vocabularies` directive was a means of instroducing symbols in special files not associated with any particular Mizar article, so that the same symbols could later be shared by multiple articles and freely overloaded by various notations. However, today's practice of the Library Committee maintaining the MML is to name these files in accordance with the name of the first article introducing a given symbol. One might obviously come to the idea that this whole set of symbols should be represented as one common resource. However, if we make such an experiment, then it is shown immediately that some of the symbols which the authors are perfectly allowed to use in specific contexts, can cause syntactic problems in others. A simple example is the use of a symbol like `[x]` for denoting a product of objects in a category [10]:

```
definition
  let C be non void non empty ProdCatStr;
```

```
  let a,b be Object of C;
  func a [x] b -> Object of C equals
  (the CatProd of C).(a,b);
  ...
end;
```

and this obviously would cause a syntactic error with passing scheme arguments like e.g. in the very formulation of the separation scheme [11]:

```
scheme
  Separation { A()-> set, P[object] } :
  ex X being set st for x holds x in X iff x in A() & P[x]
...
end;
```

With 8863 symbols in current use (1933 attributes, 4825 functors, 37 left brackets, 37 right brackets, 936 modes, 752 predicates, 175 selectors, and 168 structures)[2] the syntactic clashes would be practically unavoidable.

The rest of the directives, at least from the syntactic point of view, behave in a similar way, so it is reasonable to combine them into one common `import` directive, which should work as a macro merging 'behind the scene' several different directives. This functionality has been implemented to reflect the situation that some resources with the common name might not necessarily exist in the library. One should be able to import all the possible items from an article even if some of the imported notions were not exported to the library (e.g. most articles contain theorems, but many do not provide schemes, or new notations, registrations, etc.).

Table 1 below shows some statistics of how the directives are used in the current MML articles.

Table 1. Average number of directives per article.

Directive	Avg. number of article names
`constructors`	12.5376
`definitions`	3.41117
`equalities`	3.84251
`expansions`	2.77502
`notations`	26.879
`registrations`	15.4981
`requirements`	4.05275
`schemes`	2.56943
`theorems`	24.9581
`vocabularies`	29.3165

[2] According to the statistics provided by MMLQuery, `http://fm.uwb.edu.pl/mmlquery/fillin.php?filledfilename=statistics.mqt&argument=number+1&version=5.40.1289`

It should also be noted that for some of the Mizar environment directives the order in which their arguments appear in the article may be relevant, because of the concrete implementation limitations, e.g. `registrations` of adjectives and reductions or `equalities`. For `notations` and `definitions` the ordering is meaningful by design.

Preserving the ordering is useful to avoid errors caused by the overloading of popular symbols heavily used in the library. However, there might be cases of overly complicated environments, that the current state of the library may not allow to apply this normalization.

In each MML version, such a reference list is available in the `mml.lar` file, which lists all the processable MML articles, starting from the axiomatic `TARSKI`. So, users are generally encouraged to construct their environments in accordance with the ordering given by this list, if it can be done without difficulty.

3 Generating Standardized Environments

The proposed improvement of the user interface for accessing the library is based on introducing a new `imports` directive to the Mizar language and implementing it in the relevant software (the ACCOMMODATOR, cf. [7]). Although the new directive is primarily devised to be useful when creating new formalizations from scratch, we should also be able to re-introduce it to the available MML articles to anticipate potential problems that may result from its use and propose ready solutions. Such standardized environments for MML articles can be prepared in two simple steps with the help of a set of PERL scripts which are distributed with Mizar [16] to facilitate the analysis of MML articles' environments and their transformations.

Firstly, we can replace all the underlying directives' names into the new `import` keyword and then use an adjusted version of the `sortenv.pl` script, to turn them into one `import` directive without any repetitions among its arguments. The sorting obviously helps to keep the article's new combined environment in sync with the natural ordering of how the MML has been built from the axiomatic notions provided by `mml.lar`. Comparing to the numbers presented in Table 1, we can note that the new `imports` directives in MML articles have on average 38.5392 files as their arguments.

3.1 Examples

It turns out that if we automatically introduce the new `imports` directive to the current MML, 560 out of 1289 articles (43%) require some changes (in the environment ordering or adjusting the text proper part). We may, for example, look at selected files from the `YELLOW` series of articles that contain the introductory material formalized during the project of encoding in Mizar a handbook of continuous lattices (CCL) [17]. These articles are more or less in the middle of the `mml.lar` list, so they are quite advanced. On the other hand, they were

developed to fill gaps from several theories (topology, lattices and ordered sets), so they can simulate quite well a typical formalization.

The first article from this series which does not proof check without errors after automatically generating its environment with the new `import` directive is `YELLOW_3` [18].

The original environment declaration in the current MML version[3] looks like this:

```
environ

 vocabularies XBOOLE_0, SUBSET_1, TARSKI, ORDERS_2, WAYBEL_0, XXREAL_0,
      ZFMISC_1, RELAT_1, MCART_1, LATTICE3, RELAT_2, LATTICES, YELLOW_0,
      EQREL_1, REWRITE1, ORDINAL2, FUNCT_1, STRUCT_0, YELLOW_3;
 notations TARSKI, XBOOLE_0, ZFMISC_1, XTUPLE_0, SUBSET_1, RELAT_1, RELAT_2,
      RELSET_1, MCART_1, DOMAIN_1, FUNCT_2, BINOP_1, STRUCT_0, ORDERS_2,
      LATTICE3, YELLOW_0, WAYBEL_0;
 constructors DOMAIN_1, LATTICE3, ORDERS_3, WAYBEL_0, RELSET_1, XTUPLE_0;
 registrations XBOOLE_0, SUBSET_1, RELSET_1, STRUCT_0, LATTICE3, YELLOW_0,
      ORDERS_2, WAYBEL_0, RELAT_1, XTUPLE_0;
 requirements SUBSET, BOOLE;
 definitions LATTICE3, RELAT_2, TARSKI, WAYBEL_0, ORDERS_2;
 expansions LATTICE3, RELAT_2, WAYBEL_0, ORDERS_2;
 theorems FUNCT_1, FUNCT_2, FUNCT_5, LATTICE3, MCART_1, ORDERS_2, RELAT_1,
      RELAT_2, RELSET_1, TARSKI, WAYBEL_0, YELLOW_0, YELLOW_2, ZFMISC_1,
      XBOOLE_0, BINOP_1, XTUPLE_0;
 schemes FUNCT_7, RELAT_1;
```

And the standardised version looks as follows (please mind that the directives `vocabularies` and `requirements` stayed intact):

```
environ
 vocabularies XBOOLE_0, SUBSET_1, TARSKI, ORDERS_2, WAYBEL_0, XXREAL_0,
      ZFMISC_1, RELAT_1, MCART_1, LATTICE3, RELAT_2, LATTICES, YELLOW_0,
      EQREL_1, REWRITE1, ORDINAL2, FUNCT_1, STRUCT_0, YELLOW_3;
 requirements SUBSET, BOOLE;
 imports RELAT_1, TARSKI, XBOOLE_0, XTUPLE_0, ZFMISC_1, SUBSET_1, FUNCT_1,
      RELAT_2, RELSET_1,  MCART_1, FUNCT_2, BINOP_1, DOMAIN_1, FUNCT_5, FUNCT_7,
      STRUCT_0, LATTICE3, YELLOW_0, ORDERS_2, ORDERS_3, WAYBEL_0, YELLOW_2;
```

Several errors reported in this article with the new environment are caused by the reorganized set of imported definitions. As a solution it is enough to move the `RELAT_1` article name before `TARSKI`, so that inclusion between two relations can be proved according to the original definition for simple sets rather than using a more specialized condition in the case of relations that comes from the redefinition in `RELAT_1`. Similarly, we need to swap the order of `ORDERS_2` and `YELLOW_0`, because the author of this formalization again preferred to 'unfold' the original definition of an antisymmetric relation. In these cases the relative distance between these two files in the list was rather small, so nothing was broken by this tiny disorder. However, trying to use the same approach to solve another definition-related error would require moving `RELSET_1` before `TARSKI` in the `imports` directive and this change would have worse consequences. Namely, the Analyzer module reports an unknown functor, because the order of notations was changed at the same time, and now the `.:` function application symbol would not generate proper type information (a redefinition of the form `redefine func R .: A -> Subset of Y;` gets overloaded and then unavailable):

[3] MML ver. 5.41.1289 distributed with the compatible Mizar system ver. 8.1.06, `http://mizar.uwb.edu.pl/system/#download`

```
    thus f.(x "/\" y) = f.inf {x,y} by YELLOW_0:40
      .= inf (f.:{x,y}) by A3,A2
::>        *103
      .= inf {f.x,f.y} by A1,FUNCT_1:60
      .= f.x "/\" f.y by YELLOW_0:40;
::> 103: Unknown functor
```

So a better solution for this kind of situation is to fix the proof which causes the proof checker to complain and make use of a better suited redefinition. In this case, it is indeed a better and shorter proof:

```
theorem Th1:
  for X, Y being set, D being Subset of [:X,Y:] holds D c= [:proj1 D, proj2 D:]
proof
  let X, Y be set, D be Subset of [:X,Y:];
  let x be Element of X, y be Element of Y;
  assume
A1: [x,y] in D;
  x in proj1 D & y in proj2 D by A1,XTUPLE_0:def 12,def 13;
  hence thesis by ZFMISC_1:def 2;
end;
```

instead of the original (here commented out with the error mark `*52` indicating a wrong definition order):

```
:: theorem Th1:
::   for X, Y being set, D being Subset of [:X,Y:] holds D c= [:proj1 D, proj2 D:]
:: proof
::   let X, Y be set, D be Subset of [:X,Y:];
::   let q be object;
::   assume
:: ::>    *52
:: A1: q in D;
::   then consider x, y being object such that
::   x in X and
::   y in Y and
:: A2: q = [x,y] by ZFMISC_1:def 2;
::   x in proj1 D & y in proj2 D by A1,A2,XTUPLE_0:def 12,def 13;
::   hence thesis by A2,ZFMISC_1:def 2;
:: end;
::> 52: Invalid assumption
```

The article `YELLOW_9` [20] with a new automatically generated environment contains a few analogous errors, but can also be used to illustrate a more interesting kind of error which can happen when we merge importing directives and set a common ordering for all of them. In this case we have a space which is both topological, but also relational. The meaning of the attribute 'discrete' is different from the original and so we get:

```
registration
  cluster strict complete 1-element for TopLattice;
  existence
  proof
    take the strict reflexive 1-element discrete finite TopRelStr;
::>                                              *136
    thus thesis;
  end;
end;
::> 136: Non registered cluster
```

as well as the following syntactically correct statement, but about a different notion of 'discrete', so the final proof step is not accepted by the CHECKER:

```
registration
  let R be RelStr;
  cluster correct discrete strict for TopAugmentation of R;
  existence
  proof reconsider BB = bool the carrier of R
    as Subset-Family of R;
    set T = TopRelStr(#the carrier of R, the InternalRel of R,
BB#);
    the RelStr of R = the RelStr of T;
    then reconsider T as TopAugmentation of R by Def4;
    take T;
    T is discrete TopStruct by TDLAT_3:def 1;
    hence thesis;
::>            *4
  end;
end;
::> 4: This inference is not accepted
```

The solution, of course, is to put `TD_LAT` after `ORDERS_3` in the `imports` in order to reflect the author's original order of notations and so instead of importing the notion for relational structures:

```
definition
  let IT be RelStr;
  attr IT is discrete means
:: ORDERS_3:def 1
  the InternalRel of IT = id (the carrier of IT);
end;
```

force the system to use the version defined in topological terms for arbitrary topological structures:

```
definition
```

```
  let IT be TopStruct;
  attr IT is discrete means
:: TDLAT_3:def 1
  the topology of IT = bool the carrier of IT;
  attr IT is anti-discrete means
end;
```

Fortunately, the reordering of `TD_LAT` and `ORDERS_3` does not introduce other troublesome imports.

4 Concluding Remarks

The experimental versions of the Mizar ACCOMMODATOR (`accom` and `makeenv`) binaries precompiled for the main distribution platforms (Linux, Windows and MacOSX/Darwin), the adjusted `sortenv.pl` environment sorting script and relevant example Mizar articles can be found at the Mizar website: `http://mizar.uwb.edu.pl/~softadm/imports/`. Apart from the implementation related to adding a new importing directive, in order to accommodate successfully all the articles from the current MML some limits fixed in the current systems had to be increased, e.g. the `MaxAttrPattNbr` restricting the number of permittable attribute patterns. The new word `imports` should also be added to the list of reserved words of the language contained in the library file `mizar.dct`.

The new mechanism is inevitably going to produce some performance issues from time to time when various automation mechanisms would be overused indirectly by the users. It is, however, expected that the new mechanism could significantly help the users, especially the less experienced ones. And when their work is submitted to the MML, the best optimized environment could be restored by the Library Committee and preserved in the library.

Finally, it is worthwile to consider developing even more high-level environment importing directives. They can resemble current `requirements` by using a fixed name for a combination of carefully selected imports that enable jump start developments in specific theories, like the aforementioned continuous lattices theory, for instance.

Acknowledgement

The processing and analysis of the Mizar library has been performed using the infrastructure of the University of Bialystok High Performance Computing Center.

References

1. J. Harrison, HOL Light tutorial.
   URL `http://www.cl.cam.ac.uk/~jrh13/hol-light/tutorial.pdf`

2. Y. Bertot, P. Castéran, Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions, Texts in Theoretical Computer Science. An EATCS Series, Springer, 2004. doi:10.1007/978-3-662-07964-5.
URL http://dx.doi.org/10.1007/978-3-662-07964-5
3. T. Nipkow, L. C. Paulson, M. Wenzel, Isabelle/HOL: a proof assistant for higher-order logic, Vol. 2283, Springer Science & Business Media, 2002.
4. A. Grabowski, A. Korniłowicz, A. Naumowicz, Four decades of Mizar, Journal of Automated Reasoning 55 (3) (2015) 191–198.
5. G. Bancerek, C. Byliński, A. Grabowski, A. Korniłowicz, R. Matuszewski, A. Naumowicz, K. Pąk, J. Urban, Mizar: State-of-the-art and beyond, in: Kerber et al. [21], pp. 261–279.
6. J. Alama, mizar-items: Exploring fine-grained dependencies in the Mizar Mathematical Library, in: J. H. Davenport, W. M. Farmer, J. Urban, F. Rabe (Eds.), Intelligent Computer Mathematics - 18th Symposium, Calculemus 2011, and 10th International Conference, MKM 2011, Bertinoro, Italy, July 18-23, 2011. Proceedings, Vol. 6824 of Lecture Notes in Computer Science, Springer, 2011, pp. 276–277.
7. A. Grabowski, A. Kornilowicz, A. Naumowicz, Mizar in a nutshell, Journal of Formalized Reasoning 3 (2) (2010) 153–245.
URL http://jfr.unibo.it/article/view/1980
8. A. Naumowicz, Automating Boolean set operations in Mizar proof checking with the aid of an external SAT solver, Journal of Automated Reasoning 55 (3) (2015) 285–294.
9. A. Naumowicz, Interfacing external CA systems for Gröbner bases computation in Mizar proof checking, International Journal of Computer Mathematics 87 (1) (2010) 1–11.
10. C. Byliński, Cartesian categories, Formalized Mathematics 3 (**2**) (1992) 161–169.
URL http://fm.mizar.org/1992-3/pdf3-2/cat_4.pdf
11. Z. Trybulec, H. Święczkowska, Boolean properties of sets, Formalized Mathematics 1 (**1**) (1990) 17–23.
URL http://fm.mizar.org/1990-1/pdf1-1/boole.pdf
12. A. Naumowicz, Enhanced processing of adjectives in Mizar, Studies in Logic, Grammar and Rhetoric 18 (31) (2009) 89–Ũ101.
13. A. Korniłowicz, On rewriting rules in Mizar, Journal of Automated Reasoning 50 (2) (2013) 203–210.
14. A. Grabowski, A. Kornilowicz, C. Schwarzweller, Equality in computer proof-assistants, in: 2015 Federated Conference on Computer Science and Information Systems, FedCSIS 2015, Lódz, Poland, September 13-16, 2015, 2015, pp. 45–54. doi:10.15439/2015F229.
URL http://dx.doi.org/10.15439/2015F229
15. A. Korniłowicz, Definitional expansions in Mizar, Journal of Automated Reasoning 55 (3) (2015) 257–268.
16. A. Naumowicz, Tools for MML environment analysis, in: Kerber et al. [21], pp. 348–352.
17. G. Bancerek, P. Rudnicki, A compendium of continuous lattices in MIZAR, J. Autom. Reasoning 29 (3-4) (2002) 189–224. doi:10.1023/A:1021966832558.
URL http://dx.doi.org/10.1023/A:1021966832558
18. A. Korniłowicz, Cartesian products of relations and relational structures, Formalized Mathematics 6 (**1**) (1997) 145–152.
URL http://fm.mizar.org/1997-6/pdf6-1/yellow_3.pdf

19. A. Trybulec, Moore-Smith convergence, Formalized Mathematics 6 (**2**) (1997) 213–225.
URL http://fm.mizar.org/1997-6/pdf6-2/yellow_6.pdf
20. G. Bancerek, Bases and refinements of topologies, Formalized Mathematics 7 (**1**) (1998) 35–43.
URL http://fm.mizar.org/1998-7/pdf7-1/yellow_9.pdf
21. M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, V. Sorge (Eds.), Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings, Vol. 9150 of Lecture Notes in Computer Science, Springer, 2015.