Towards Standardized Mizar Environments

Adam Naumowicz

Institute of Informatics
University of Białystok, Poland
adamn@mizar.org

ISAT 2017, Szklarska Poręba, September 18, 2017

What is Mizar ?

- Mizar is a system for encoding and proof-checking mathematics invented by Andrzej Trybulec (†2013) and developed since 1970s.
- Its language tries to mimic standard mathematical practice.
- Its verification engine is designed to preserve human understanding of proof steps.
- It is being used to build a centralized library of formalized mathematical knowledge based on simple axioms (of set theory) and special focus on reusability - Mizar Mathematical Library (MML).

Formalization Environments

- 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.

Various Ways of Importing Knowledge in Proof Assistants

- HOL Light has the import handled by its metalanguage OCaml's `#use` and the `needs` directive
- `Require [Import|Export]` in Coq
- `imports` directive in Isabelle/Isar
- ...
- In current Mizar there is a set of several fine-grained importing directives with different roles and semantics.

Mizar Importing Mechanism

- Mizar importing mechanism comes from the times of rather scarce computer resources (RAM, storage, computational power).
  - It was designed as a practical solution to cope with these limitations.
- It still works reasonably well nowadays when the complexity of the formalized developments grows.
  - Most of the current complex formalization developments are still processed in seconds or minutes rather than hours.
  - Even more fine-grained access to every particular definition and proof can sometimes be desirable (e.g. `mizar-items` project for reconstructing the prerequisites of theorems in the spirit of reverse mathematics).

Formalization Reusability

- The level of reusability in other large formalization libraries is currently significantly smaller than in the Mizar library.
- Isabelle-based Archive of Formal Proofs (as presented at CICM2015):
  - 106 out of its 215 articles were isolated nodes of the imports graph, i.e. they were not related to other articles in the library.
  - Among the articles that had some non-trivial dependence, the maximal number of reused articles in one article was equal to 4, and the maximal number of articles directly depending on some other article reached 9.
- MML ver. 5.41.1289:
  - Only one isolated node (for technical reasons preserved in the library, (article SCHEMS_1), There is an article which imports data from 121 other articles (JORDAN)
  - The elementary properties of subsets from SUBSET are imported in as many as 1250 articles.

Mizar Importing Directives

- The directives are:
  - `vocabularies`,
  - `notations`,
  - `constructors`,
  - `registrations`,
  - `requirements`,
  - `definitions`,
  - `equalities`,
  - `expansions`,
  - `theorems`,
  - `schemes`.
- Apart from `requirements` and `vocabularies`, each directive contains a list of (usually a couple of dozens) article names whose data should be imported to give a certain meaning to the newly developed formalization.

Mizar Requirements

- The `requirements` directive accepts as its arguments specific names which do not directly correspond to any formalization article
  - It provides means to switch on special processing for selected highly-used notions,
  - e.g. `BOOLE` for automating Boolean operations on sets, or `ARITHM` for automating the arithmetic of real and complex numbers.
- The `vocabularies` directive was originally a means for introducing symbols in special files not associated with any particular Mizar article.
  - The same symbols can later be shared by multiple articles and freely overloaded by various notations.
  - 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.

Why Not Import All Vocabulary Symbols?

- Consider the use of a symbol like `[x]` for denoting a product of objects in a category:

```
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;
```

- This would cause a syntactic error with passing scheme arguments like e.g. in the very formulation of the separation scheme:

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

Mizar Vocabularies

- The `vocabularies` directive allows to share the same symbols by many (sometimes) very different definitions.
- Importing all their respective data when we just need to use a certain symbol would be completely unreasonable.
- We do not want the lexical analysis lose its flexibility by requiring that all symbols introduced in some article become at that point reserved for all further library uses.
- 8863 symbols in current use (1933 attributes, 4825 functors, 37 left brackets, 37 right brackets, 936 modes, 752 predicates, 175 selectors, and 168 structures).

Other (Standard) Mizar Directives

- From a syntactic point of view, the rest of the directives behave in a similar way.
- It is reasonable to combine them into one common `import` directive
  - It should work as a macro merging 'behind the scene' several different directives.
  - 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.).

Average Number of Directives Per Article

Directive	Avg. number of article names
constructors	12.53760
definitions	3.41117
equalities	3.84251
expansions	2.77502
notations	26.87900
registrations	15.49810
requirements	4.05275
schemes	2.56943
theorems	24.95810
vocabularies	29.31650

The Order of Mizar Imports

- For some of the Mizar environment directives the order in which their arguments appear in the article may be relevant.
  - 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.
- 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`.
- Users are generally encouraged to construct their environments in accordance with the ordering given by this list, if it can be done without difficulty.

Introducing the New Imports Directive

- A new `imports` directive has been implemented the Accommodator.
- It is primarily devised to be useful when creating new formalizations from scratch.
- Re-introducing it to the available MML articles helps to anticipate potential problems that may result from its use and propose ready solutions.

Preparing Standardized Environments

1. Replace all the underlying directives' names with the new `import` keyword.
2. 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`.
- The new `imports` directives in MML articles have on average 38.5392 files as their arguments.

Re-introducing Imports

- 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,
  - adjusting the text proper part.

Examples

- The YELLOW series of articles contain the introductory material formalized during the project of encoding in Mizar a handbook of continuous lattices (CCL).
  - These articles are more or less in the middle of the `mml.lar` list, so they are quite advanced.
  - 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.

Examples: YELLOW_3

- The original environment declaration in the current MML 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;
```

- 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;
```

Examples: YELLOW_3

- Several errors reported in this article with the new environment are caused by the reorganized set of imported definitions.
- Solution:
  1. 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`.
  2. Similarly, 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.

Examples: YELLOW_3

- 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.
- 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):

```
    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
```

Examples: YELLOW_3

- 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;
```

- 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
```

Examples: YELLOW_9

- The article YELLOW_9 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
```

Examples: YELLOW_9

- The following is a 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
```

Examples: YELLOW_9

- The solution, 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.

Conclusions and Future Work

- 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/.
- 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 can 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.

Conclusions and Future Work

- It is also worthwhile to consider developing even more high-level environment importing directives.
- The presented experiment sheds some light on how the granularity of import mechanisms could be further developed in Mizar to better serve the users.
- It also shows the users that it is beneficial to organize the file structure of their newly formalized theories in a more fine-grained way to avoid import conflicts during the reuse of their data.
- The research carried out over the Mizar library allows to anticipate similar potential problems with reorganizing the rich dependence structure of other formal mathematical libraries.