A New Export of the Mizar Mathematical Library

Colin Rothgang[1(✉)], Artur Korniłowicz[2], and Florian Rabe[3]

[1] Mathematics, TU Berlin, Berlin, Germany
colin.rothgang@posteo.net
[2] Institute of Computer Science, Bialystok, Poland
[3] Computer Science, FAU Erlangen-Nürnberg, Erlangen, Germany

Abstract. The Mizar Mathematical Library (MML) is a prime target of library exports, i.e., translations of proof assistant libraries that make the libraries available to knowledge management systems or other deduction systems. The MML has been exported multiple times in the past, including our own export from Mizar to OMDoc done in 2011. But the exporters tend to be very difficult and expensive to maintain.

We present a complete reimplementation of our previous export. It incorporates many lessons learned and leverages improvements made both on the Mizar and the OMDoc side.

1 Introduction

The interoperability of proof assistants and the integration of their libraries is a longstanding goal in theorem proving. One of the biggest prizes here is the Mizar Mathematical Library (MML) [BBG+18]. It has been exported multiple times [Urb03, IKRU13] and other efforts are ongoing [KP19]. Mizar stores all kernel data structures in externally readable XML files [Urb05] so that exports are reduced to interpreting the XML files. However, Mizar generates about a dozen XML files per MML article, and these use ad-hoc, under-documented, and evolving XML schemas that are as complex as Mizar's feature-rich language and contain many parts that are only needed internally.

This was a major difficulty in our first translation of the MML to OMDoc [IKRU13], an XML-based representation format for mathematical knowledge and made the translation, while successful, prohibitively difficult to maintain. In the ten years since then, two things have changed. The Mizar XML data structures have been heavily improved, both for internal reasons and in response to the woes of translation developers. And we have developed much better routines for prover library translations [KR20]. The present system description redoes the export from scratch.

Like [IKRU13], we manually formalize the Mizar logic as a theory M in a logical framework of the LF family, which is realized in MMT [RK13]. Then we generate one MMT theory relative to M for every article in the MML. All these MMT theories are stored in OMDoc format [Koh06] and are available online. During the export, we aim at both preserving all Mizar features exactly as they are (as opposed to implementing logically complex feature eliminations) and simplifying the language by reducing features to the primitives provided by the MMT framework.

© Springer Nature Switzerland AG 2021
F. Kamareddine and C. Sacerdoti Coen (Eds.): CICM 2021, LNAI 12833, pp. 205–210, 2021.
https://doi.org/10.1007/978-3-030-81097-9_17

Besides rejuvenating the export, we go beyond [IKRU13] in multiple ways: (i) We implement the relevant parts of Mizar's XML schema as a set of Scala data types, from which the parser is automatically generated, thus massively simplifying the maintenance of the translation. (ii) Our translation is almost entirely context-free (every file can be exported without processing its dependencies), which is critical for scalability of the export, and we identify the last remaining context-sensitivity issues in the XML schemas. (iii) We represent Mizar's rich set of declarations as MMT patterns in the sense of [HR15] and Mizar's structures (akin to record types) as an MMT structural feature in the sense of [MRRK20]. (iv) We cover many recondite Mizar features that were not handled well in [IKRU13] such as redefinitions.

2 Design

2.1 Formalizing the Mizar Logic

At the object-level, we formalize Mizar's softly-typed set theory in a logical framework in MMT. We omit technical details that are already part of [IKRU13] and only point out that we use LF-style HOAS with {_}_ and [_]_ representing Π and λ. At the declaration-level, we represent Mizar's many conservative extension principles such as definitions and registrations as MMT patterns [HKR12] and structural features [MRRK20]. The formalization is available at https://gl.mathhub.info/MMT/LATIN2/.

For example, Mizar's direct partial predicate definitions are formalized as the MMT pattern below. Its header takes natural numbers n (the arity of the new predicate pred) and m (the number of cases in its definition), a list argTps of n argument types (each potentially depending on the other arguments), m cases (each consisting of a condition $cases_i$ on the n arguments and the resulting predicate $\mathrm{caseRes}_i$), a default result defRes if no case applies, and a proof cons that the results of cases agree when their conditions overlap (we omit the definition of consDirectPredDef). These are the argument given in Mizar and exported in the XML files. The body of the pattern contains the elaboration performed by Mizar: the n-ary predicate pred and its defining axiom means. Typical for all pattern is the heavy use of MMT's support for flexary operators such as conjunction and mapping over argument sequences.

```
pattern directPartPredDef(n : NAT, m : NAT, argTps : ($term^n$ → tp)$^n$,
    cases : ($term^n$ → prop)$^m$, caseRes : ($term^n$ → prop)$^m$, defRes : $term^n$ → prop
    cons : consDirectPredDef n argTps m cases caseRes)      =
  pred : $term^n$ → prop
  consistency = cons
  means : {x : $term^n$} ⟨ ⊢ x.i ⦂ (argTps.i) x | i : n⟩ →
    ⊢ nary_and m ⟨ (cases.j) x  ⇒  (pred x)  ⇔  (caseRes.j) x | j : m ⟩
    ∧ nary_and m ⟨ ¬ (cases.k) x | k : m ⟩  ⇒ pred x  ⇔ defRes x
```

2.2 Exporting the MML as XML

The Mizar verifier creates several XML files per source file that store information of the various processing phases. The most important are Weakly Strict Mizar (WSM, `.wsx` files) [BA12,NP16], which contain the syntax trees of statically analyzed articles and More Strict Mizar (MSM, `.msx` files), which extends WSM with the resolution of variables, constants, and labels. To additionally keep the original syntax, Even more Strict Mizar (EMSM, `.msx` files) is being developed, which adds semantic information about used *constructors* and Mizar *patterns* (which store the used format, constructor, argument types, and positions of visible arguments for a definition).

For example, the definition of subset below contains the formula x in Y, which results in EMSM in the XML fragment underneath. Here the attribute spelling gives the original syntax. The other blue-highlighted attributes are what we can use to form identifiers in OMDoc for the three different kinds of references: the constant in defined in the MML, and the kinds of bound variables Y (bound by the declaration) and x (bound by a quantifier). The other attributes (nr, formatnr, etc.) represent internally used numbers that we ignore. The decision which attributes to use/ignore requires a Mizar expert but is now documented in our export.

```
definition let X,Y; pred X c= Y means for x being object holds x in X implies x in Y;
reflexivity; end;
```

```
<Relation−Formula nr="3" formatnr="6" patternnr="3" absolutepatternMMLId=
  "HIDDEN:3" leftargscount="1" spelling="in" sort="Relation-Formula"
  constrnr="2" absoluteconstrMMLId="HIDDEN:2" originalnr="0" position="72\47">
<Arguments>
<Simple−Term idnr="2" spelling="x" position="72\44" origin="BoundVar"
  sort="BoundVar" serialnr="29" varnr="1"/>
<Simple−Term idnr="9" spelling="Y" position="72\49" origin="ReservedVar"
  sort="Constant" serialnr="8" varnr="2"/></Arguments></Relation-Formula>
```

2.3 Reading the XML into Scala Classes

Contrary to other parser generators, which generate the source code of the classes and the parser from a grammar, we directly implement the classes in Scala and then generate the parser. For example, the (heavily simplified) classes below are used to pick out the relevant attributes from the Simple_−Term element visible above. Expression is the abstract class of all Mizar expressions, and LocalConstAttr is an auxiliary class that groups XML attributes that often occur together.

```
abstract class MizTerm extends Expression
case class Simple_Term(serialnr: Int, spelling:String, sort:String) extends MizTerm
case class Arguments(_children:List[MizTerm])
```

This is sufficient to generate the XML parser using Scala reflection. This has the advantage that the Scala classes, which are what the translation developer interacts with primarily in the next step, are much more easily maintainable, can be better documented than generated code, and can be manually tweaked better to be practical.

These classes are available at https://github.com/UniFormal/MMT/ in the package `info.kwarc.mmt.mizar`. Note that this XML parsing step is independent of MMT. Thus, other developers can easily reuse these classes and our XML parser as a starting point for translations into other target languages.

2.4 Translating the Scala Classes to MMT

The logical heart of the translation is now isolated in an inductive function that traverses the Scala classes holding the MML XML and producing corresponding MMT classes. This happens in memory, and MMT's existing emitters for OMDoc and MMT source syntax can be used out of the box. Critically, it requires handling all idiosyncrasies of the Mizar language. The resulting export of the MML is available at https://gl.mathhub.info/Mizar/MML.

We translate each Mizar **article** to an MMT theory (relative to those from Sect. 2.1) that contains *include* declarations for all Mizar articles it depends on. Each **theorems** is translated to a single MMT declaration whose type gives the claim and whose definiens the proof. Each scheme, functor/predicate/mode, and attribute **definitions** as well as **synonyms/antonyms** and **registrations** are translated into instances of the corresponding patterns mentioned in Sect. 2.1. Where applicable, they are followed by declarations stating and proving the specially treated properties such as reflexivity for binary predicates. **Redefinitions** are translated into fresh constants with a new definition; if only the type is changed and no new definiens is provided, we synthesize a definition by applying the original constant to the corresponding arguments (this works, as the new types are required to be subtypes). **Structure** definitions (record types) are translated using an MMT structural feature in the style of [MRRK20] that reimplements in MMT Mizar's conservative extension principle for adding named record types. Mizar's forgetful functors between structures become record subtyping.

Proofs, which Mizar does not store entirely anyway, are translated only partially by using a special constant for a proof oracle: it takes a claim and a number of references to used theorems and returns a proof of the claim. This way proof dependencies are preserved in the export.

Improving on [IKRU13], our translation covers correctness conditions and properties of definitions, all registrations ([IKRU13] covered only existential registrations), forgetful-functors between structures, and partial proofs.

For example, consider the definition of the subset predicate from Sect. 2.2, which uses $n = 2$ arguments, whose types are just set and do not depend on other arguments, $m = 0$ cases, and one default case for the actual definition. Its name R1R1 is build from a character characterizing the kind of declaration (R) and the article-scoped counters contained in the absolutepatternMMLId and absoluteconstrMMLId, which yields a unique identifier within the article. It is translated to the following instance of the MMT pattern directPartPredDef from Sect. 2.1 (the formula $\text{true} \wedge \text{true} \Rightarrow \_$ is deliberately not simplified, to emphasize how it is derived by elaborating the directPartPredDef pattern):

```
instance tarski:R1R1 ?MizarPatterns/directPartPredDef(2 0 ⟨([x : term^2] set), ([x : term^2] set)⟩
   ⟨⟩ ⟨⟩ ([x : term^2] for term [x/BV/29 : term] x/BV/29 in x.0 ⇒ x/BV/29 in x.1)
   (proof omitted))
//elaborates to
pred : term^2 → prop
consistency = (proof omitted)
means : {x : term^2} ⊢ x.0 ⸱⸱ set →⊢ x.1 ⸱⸱ set →⊢ true ∧ true
   ⇒ (tarski:R1R1/pred x) ⇔ (for term [x/BV/29 : term] x/BV/29 in x.0 ⇒ x/BV/29 in x.1)
```

The translation is **context-free** except for a few cases where the EMSM files do not quite contain enough information yet—in those cases some static analysis of Mizar must be reimplemented in MMT and therefore the depended-upon articles must have been processed already:

- Some functor redefinitions and functorial registrations require type inference of the return type
- The arity of the original declaration of a redefinition without definiens must be determined.

The Mizar developers plan to address this issue with a new set of files in a new extension of EMSM.

3 Conclusion and Future Work

Format	Size	Gen. time
MML	100 MB	-
XML	4.7 GB	15 min
MMT (zipped OMDoc)	18 MB	2 h
MMT (text syntax)	200 MB	30 h

We have presented a thorough overhaul of the 10-year old export of the MML into OMDoc, leveraging all lessons learned and improvements made since then. Our export covers the entire MML with the only exception being the partial translation of proofs. The table on the right gives an overview of the sizes and generation times of the digital artifacts, all of which are available online as referenced throughout the paper. The export has so far been run only on simple hardware, and we expect shorter times when parallelizing on a server as soon as all issues of context-sensitivity have been removed. The generation time of the Mizar XML is so short because it is parallelized and Mizar only needs to resolve identifiers, which does not require verifying the proofs. The generation time of the MMT text syntax is excessive due to a scalability issue in MMT that was uncovered by the present export; it is unrelated to the Mizar export and will be fixed in a future release. The sizes of the Mizar and the MMT text are not directly comparable: the latter lacks full proofs, but includes some longer generated variable names, includes instances with their elaborations, and uses a less optimized syntax for conciseness and readability.

The XML produced by Mizar has much higher quality, the representation uses modern MMT feature for declaratively mimicking Mizars's highly idiosyncratic language, and the export implementation is substantially more maintainable, easier to use, and scalable. This provides promising evidence that investments into proof assistant library export workflows (albeit costly ones at glacial pace) are putting library translations and the thus-enabled system integrations ever more feasible.

References

BA12. Bylinski, C., Alama, J.: New developments in parsing Mizar. In: Jeuring, J., Campbell, J.A., Carette, J., Dos Reis, G., Sojka, P., Wenzel, M., Sorge, V. (eds.) CICM 2012. LNCS (LNAI), vol. 7362, pp. 427–431. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31374-5_30

BBG+18. Bancerek, G., et al.: The role of the Mizar Mathematical Library for interactive proof development in Mizar. J. Autom. Reason. **61**(1), 9–32 (2018). https://doi.org/10.1007/s10817-017-9440-6

HKR12. Horozal, F., Kohlhase, M., Rabe, F.: Extending MKM formats at the statement level. In: Jeuring, J., et al. (eds.) CICM 2012. LNCS (LNAI), vol. 7362, pp. 65–80. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31374-5_5

HR15. Horozal, F., Rabe, F.: Formal logic definitions for interchange languages. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) CICM 2015. LNCS (LNAI), vol. 9150, pp. 171–186. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20615-8_11

IKRU13. Iancu, M., Kohlhase, M., Rabe, F., Urban, J.: The Mizar Mathematical Library in OMDoc: translation and applications. J. Autom. Reason. **50**(2), 191–202 (2013). https://doi.org/10.1007/s10817-012-9271-4

Koh06. Kohlhase, M.: OMDoc – An Open Markup Format for Mathematical Documents [version 1.2]. LNCS (LNAI), vol. 4180. Springer, Heidelberg (2006). https://doi.org/10.1007/11826095

KP19. Kaliszyk, C., Pak, K.: Semantics of Mizar as an Isabelle object logic. J. Autom. Reason. **63**(3), 557–595 (2019). https://doi.org/10.1007/s10817-018-9479-z

KR20. Kohlhase, M., Rabe, F.: Experiences from exporting major proof assistant libraries (2020). see https://kwarc.info/people/frabe/Research/KR_oafexp_20.pdf

MRRK20. Müller, D., Rabe, F., Rothgang, C., Kohlhase, M.: Representing structural language features in formal meta-languages. In: Benzmüller, C., Miller, B. (eds.) CICM 2020. LNCS (LNAI), vol. 12236, pp. 206–221. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53518-6_13

NP16. Naumowicz, A., Piliszek, R.: Accessing the Mizar library with a weakly strict Mizar parser. In: Kohlhase, M., Johansson, M., Miller, B., de de Moura, L., Tompa, F. (eds.) CICM 2016. LNCS (LNAI), vol. 9791, pp. 77–82. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42547-4_6

RK13. Rabe, F., Kohlhase, M.: A scalable module system. Inf. Comput. **230**(1), 1–54 (2013)

Urb03. Urban, J.: Translating Mizar for first order theorem provers. In: Asperti, A., Buchberger, B., Davenport, J.H. (eds.) MKM 2003. LNCS, vol. 2594, pp. 203–215. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36469-2_16

Urb05. Urban, J.: XML-izing Mizar: making semantic processing and presentation of MML easy. In: Kohlhase, M. (ed.) MKM 2005. LNCS (LNAI), vol. 3863, pp. 346–360. Springer, Heidelberg (2006). https://doi.org/10.1007/11618027_23