Syntactic-Semantic Form of Mizar Articles

Czesław Byliński
Computer Networks Section, University of Bialystok, Poland

Artur Korniłowicz[1]
Institute of Computer Science, University of Bialystok, Poland

Adam Naumowicz
Institute of Computer Science, University of Bialystok, Poland

Abstract

Mizar Mathematical Library is most appreciated for the wealth of mathematical knowledge it contains. However, accessing this publicly available huge corpus of formalized data is not straightforward due to the complexity of the underlying Mizar language, which has been designed to resemble informal mathematical papers. For this reason, most systems exploring the library are based on an internal XML representation format used by semantic modules of Mizar. This representation is easily accessible, but it lacks certain syntactic information available only in the original human-readable Mizar source files. In this paper we propose a new XML-based format which combines both syntactic and semantic data. It is intended to facilitate various applications of the Mizar library requiring fullest possible information to be retrieved from the formalization files.

2012 ACM Subject Classification Information systems → Extensible Markup Language (XML)

Keywords and phrases Mizar system, mathematical knowledge representation, XML representation

Digital Object Identifier 10.4230/LIPIcs.ITP.2021.11

Supplementary Material A collection of prototype hyper-linked Mizar articles generated from the new ESM syntactic-semantic format can be found at
Dataset: `http://alioth.uwb.edu.pl/~artur/mmlesx/`

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

1 Introduction

Since the beginning of the Mizar project [5], the *verifier* has been the main computer program of the whole system. Its function is to check user-provided input files (called "Mizar articles") in terms of syntactic correctness and logical validity of mathematical content encoded in the Mizar language. The application has been designed analogously to building compilers, so it has a lexical and syntactic analysis modules as well as a module for performing the semantic analysis of the source text. But of course the Mizar verifier does not generate any executable code. Instead, the last module confirms that a given article does not contain errors, or otherwise reports a list of errors found in the text. Hence, the absence of errors in the result of text analysis means the correctness of the text in terms of syntactic, semantic and logical content. In the initial Mizar implementations, this was the sole result of running the verifier, i.e. the tool provided only the information about encountered errors in the form of a text listing file.

The development of the Mizar language conducted under constant leadership of Andrzej Trybulec was reinforced by the experience related to the creation of new Mizar texts as well as experiments with various limited versions of Mizar (e.g. Mizar MSE, Mizar HPF) [12]. At

[1] Corresponding author

the same time, the source code of Mizar verifiers evolved as a result of implementations on subsequent computers. Notably, Mizar 2 (the predecessor of current Mizar) was implemented on one of Polish third-generation computers, Odra 1305, using the Pascal compiler available for ICL 1900 machines. This version of the compiler was a port of Pascal P2 from the CDC 6000 machine to ICL 1900. At that time, the Mizar language was designed in such a way, so that the parsing, semantic analysis, and logical correctness procedures could be implemented as a one-pass program on the mainframe computer. The coding made use of the "top-down" approach with main algorithms based on recursive procedures. Then, Mizar 2 code was transferred from Odra 1305 to IBM 360 using Pascal P8000 with small code changes and later formed the base of implementing Mizar on the PDP-11 minicomputer. Due to the memory limitations of that machine, a one-pass implementation was not possible and therefore the verifier was divided into a series of passes implementing the subsequent stages of Mizar text processing and analysis. This change has become a permanent feature of the system since then. Initially there were seven passes which covered tokenization, syntax analysis, identifier analysis, semantic analysis of Mizar's linguistic structures, checking proof structure, inference correctness and schematization. The division into the passes was due to both content-related and technical reasons. The information about subsequent results of the analysis between the passes was transmitted via text files. The syntax of these files was strictly technical, it was only about providing information necessary for further analysis. The communication files were treated as temporary working files, generated only for the duration of the verifier's work. The final result of the text analysis was realized a bit differently than in Mizar 2. Namely, one error file was created, to which specific errors detected by individual passes were added. When the Mizar verifier was ported from PDP-11 to the IBM PC machines using the Turbo Pascal compiler, the number of passes was limited to four. At that time, the Mizar script still contained an axiomatic part describing the mathematical theory necessary to formulate new definitions and prove theorems.

A next significant change in the Mizar implementation was related to the creation of the Mizar Mathematical Library (MML) [4]. The Mizar script took the modern form of current Mizar articles, containing the so-called environment part (MML references) and the actual text. The verifier, or actually additional programs based on the code of the verifier, have been used to create library files describing all the notions, definitions and theorems introduced and proven in a given article. Current MML provides an interrelated library built from axiomatic foundations (Tarski-Grothendieck set theory) in which all derived facts are positively checked by the Mizar verifier and can be included in the MML only under this condition. Further development of Mizar software and the MML are now closely inter-related. Each new version of the Mizar software must be MML compliant. And on the other hand, after any language changes, MML must be refactored to be compatible with the new Mizar system.

At some stage of work on the implementation of the Mizar verifier (around 2004) and, more broadly, the Mizar system, the structure of MML database files and intermediate files transmitting information between the verifier's modules was changed. Instead of specific internal formats available only from within the Pascal (Delphi and Free Pascal) implementation of the system, a new XML-based form was introduced [17]. Initially it was only a technical change, but soon the more easily accessible information about Mizar articles started to be used by several external applications (c.f. [8, 18, 7]). Most of these applications have utilized the XML file carrying the information passed to the Mizar inference checker and schematizer modules. However, the disadvantage of these files is that they contain already pre-processed semantic information needed for the checker, which only partially allows reconstructing the original content of the article needed for some applications. As such they are not always suitable for use independently of the Mizar system, e.g. for the needs of formal systems other than Mizar. In recent years, several attempts have been made to use the MML for other proof systems (c.f. [9, 20, 10]). So there emerged a need to provide readily-available information about the content of the MML without the necessity of analyzing Mizar articles. The idea was to develop a format describing Mizar articles in which the formalizations contained in the MML could become accessible by any semantic mathematics databases or other formal systems. Consequently, the current versions of the Mizar verifier, in addition to the basic user feedback, i.e. the confirmation that a processed article is correct, also generate the description of the article in a syntactic form with the associated semantics as XML files. In 2012, the creation of files describing the syntactic form of Mizar articles was re-organized. The corresponding files are generated by the two initial modules of the Mizar verifier: the parser which creates a description of the syntactic tree of the analyzed article in a file technically named WSM (Weakly Strict Mizar) [7, 14], and an identification module which creates an MSM file (More Strict Mizar). MSM files are descriptions of the syntax of the Mizar article with additional information about the identifiers used: sentence and variable identifiers. However, they do not contain information about the exact constructors used and being defined. Moreover, they do not provide unambiguous information that the Mizar system uses to process the whole MML.

Recently (2020), a new variant of the Mizar verifier's Analyzer module has been implemented. This module is responsible for the semantic analysis of the Mizar language structure. Now it is possible to generate semantic information, while preserving the syntactic information used in the original Mizar source text. In previous versions of the Analyzer module, all expressions were at that point transformed into the form of so called "semantic correlates" used by the Checker module. This resulted in the loss of information about their syntactic form and rendered the exact reconstruction of the original syntax impossible, because the transformation is irreversible [15]. The most recent implementation of the Analyzer preserves enough information to create an article description in the form of a syntactic tree extended with associated semantic data. The new data format was named ESM (Even more Strict Mizar). The corresponding file generated by the Mizar verifier contains an exact description of the syntactic structure of the article in connection with the resolved semantic information. The disambiguation concerns all entered symbols and the semantics of definitions, expressions and theorems introduced and used in the article. Moreover, unlike the previously available data formats storing only the information necessary for proof checking by the Mizar verifier, the ESM representation is enriched with absolute references to database items (within a particular MML version), which can be understood regardless of the local environment of a given article. This information is eventually suitable for direct use by any formal systems independently of the Mizar software.

2 Even more Strict Mizar (ESM)

In this section we present the extension of previously available WSM and MSM data representation formats (generated by the Mizar verifier as intermediate files with corresponding `.wsx` and `.msx` filename extensions). Although the ESM new data format gives access to information currently stored in the XML (`.xml`) files generated by the Mizar analyzer module, it does not replace the old representation completely, since the latter is still used as the main input for the checker pass. The advantage of ESM, however, is that this representation of Mizar texts can be used independently from the dedicated Mizar proof checking software.

Table 1 Constructors (C) and patterns (P): + means introduction of a new constructor/pattern.

	Predicate		Attribute		Mode		Functor	
	C	P	C	P	C	P	C	P
definition	+	+	+	+	+	+	+	+
redefinition of definiens		+		+		+		+
redefinition of result type	n/a	n/a	n/a	n/a	+	+	+	+
redefinition with properties	+	+	n/a	n/a	n/a	n/a	+	+
synonym		+		+		+		+
antonym		+		+	n/a	n/a	n/a	n/a

2.1 Mizar definitions

The Mizar language allows users to define mathematical notions of several basic categories: predicates, adjectives, types, operations and structures. Using the Mizar terminology, various aspects of all these definitions are internally represented either as *formats*, *constructors*, or *patterns*. Formats represent syntactic information about the kind of symbol used in the definition together with the number and position of arguments. Constructors, on the other hand, provide a numbering system for representing the semantics of mathematical objects. The unique numbering scheme is provided independently within each category of defined notions. And finally patterns represent joint information about the used format, constructor, types of all arguments, and positions of visible arguments for a given definition. The distinction between constructors and patterns is necessary to accommodate various linguistic features like synonyms or redefinitions available in the Mizar language which offer different contextual ways of representing semantically equal objects. In principle, there are more patterns than constructors since not all definitions introduce new constructors, see Table 1. In the current Mizar implementation, the information about all available and newly defined notions is scattered across multiple intermediate XML-based resource files. The next section presents basic technical details concerning these files (identified by their respective filename extensions by Mizar utilities) necessary for developing software which makes use of all definition-related information.

2.2 Intermediate file types

First of all, new symbols introduced to denote concepts defined in a Mizar article are collected in the `.dcx` file, which also includes information about all Mizar reserved words used in the processed article. Each symbol is represented by its category (`kind`), number (`nr`), and spelling (`name`). It should be noted that symbols of different categories are numbered separately.

After the article is parsed, the formats are collected in `.frx` files. Each format is represented by `kind`, format number (`nr`), symbol number (`symbolnr`, computable from `.dcx`), number of visible arguments (`argnr`), including the number of left arguments (`leftargnr`). The number of right arguments can obviously be computed as the difference between the values of `argnr` and `leftargnr`. Notably, the numbering is continuous here, i.e. it is common to all kinds of definitions introducing the formats. Moreover, it should be noted that not all definitions introduce new formats – if the notation signatures are repeated, only one copy of the format is collected. Specific binding force of operations (if different from the default value) is also included in the format files, but it is not used to distinguish between formats.

The internal constructor descriptions are scattered across two files. Firstly, constructors imported from the database are collected in `.aco` files which consist of two sections. The first section, marked with the `<SignatureWithCounts>` element, contains incremental sums of constructors added by successive files imported from the database. By performing appropriate calculations on these sums, it is possible to reconstruct the name of the article and the number of the definition introducing a given constructor in later procedures. That section is followed by a list with descriptions of all imported constructors. Such a description contains a number of attributes representing the constructor's category (`kind`), the name of the article (`aid`) in which the constructor was introduced, the number (`nr`) in this article, and the relative number (`relnr`) of a given constructor in relation to all constructors available in a given article, the types of arguments, result types, and the indication whether a given constructor introduces any property. Constructors are numbered in their respective categories, i.e., mode constructors separately from predicate constructors, and so on. Constructors of concepts introduced in a given article are collected in `.xml` files. Their descriptions are consistent with those of imported constructors.

The most extensive description of defined objects is represented by patterns. Patterns imported from the database are stored in `.eno` files and their content is based on the following attributes:

- `kind` – category of the pattern
- `nr` – number within one imported file
- `aid` – imported article file name
- `formatnr` – number of used format
- `constrkind` – kind of used constructor
- `constrnr` – number of used constructor
- `relnr` – relative number w.r.t. a category

With the new ESM language representation, newly defined patterns are stored by the verifier in `.esx` files (in previous releases of the Mizar system, they were stored in `.xml` files). Their extended description is based on the following attributes:

- `spelling` – spelling of the token (retrieved from the format)
- `position` – position of the token
- `formatdes` – description of used format (retrieved from the format)
- `formatnr` – number of used format
- `patternnr` – number of the pattern within a category w.r.t. the environment
- `absolutepatternMMLId` – unique identifier of the pattern w.r.t. the whole MML, but within a pattern category, e.g., "FUNCT_2:4" where FUNCT_2 is an MML article identifier
- `origpatternnr` – number of the original (being redefined) pattern within a category w.r.t. the environment in redefinitions
- `absoluteorigpatternMMLId` – unique identifier of the original (being redefined) pattern w.r.t. the whole MML, but within groups, in redefinitions
- `constr` – category (kind) and number of the constructor w.r.t. the environment, e.g., "V4" where V represents the category of adjectives
- `absoluteconstrMMLId` – unique identifier of the constructor w.r.t. the whole MML, but within a given category
- `origconstrnr` – number of the original (being redefined) constructor w.r.t. the environment in redefinitions
- `absoluteorigconstrMMLId` – unique identifier of the original (being redefined) constructor w.r.t. the whole MML, but within a given category, in redefinitions.

To summarize, Table 2 shows the location of definition-related data in Mizar resource files.

Table 2 Location of definitional data in resource files.

Resource	Imported	Defined
symbol	`.dcx`	`.dcx`
format	`.frx`	`.frx`
constructor	`.aco`	`.xml`
pattern	`.eno`	`.xml` (also `.esx`)

Below we present a series of snippets of Mizar code and corresponding ESM representations (underlined attributes with values in bold face extend the information inherited from WSM and MSM formats). We start with a definition of the first projection (selecting first elements of contained pairs). This operation defined for an arbitrary set in the MML article `XTUPLE_0` is presented in Listing 1.

Listing 1 Definition of the first projection

```
definition
  let X be set;
  func proj1 X -> set means
:: XTUPLE_0:def 12
  x in it iff ex y st [x,y] in X;
  correctness;
end;
```

The underlying ESM representation of the definition is given in Listing 2. The `Standard-Type` element within `Loci-Declaration` corresponds to the type of the argument, i.e. “set” introduced in the MML article `HIDDEN`. Moreover, the `Type-Specification` element indicates that in this general context the result type of the operation is also a set.

Listing 2 ESM representation of the first projection (definiens is elided)

```
<Block kind="Definitional-Block" position="192\10" endposition="202\3">
<Item kind="Loci-Declaration" position="193\5" endposition="193\14">
<Loci-Declaration>
 <Qualified-Segments>
  <Explicitly-Qualified-Segment position="193\7">
   <Variables>
    <Variable idnr="25" spelling="X" position="193\7" kind="Constant"
     serialnr="54" varnr="1"/>
   </Variables>
   <Standard-Type nr="1" formatnr="3" patternnr="2" absolutepatternMMLId="HIDDEN:2"
     spelling="set" sort="Mode" constrnr="2" absoluteconstrMMLId="HIDDEN:2"
     originalnr="0" position="193\14">
    <Arguments/>
   </Standard-Type>
  </Explicitly-Qualified-Segment>
 </Qualified-Segments>
</Loci-Declaration>
</Item>
<Item kind="Functor-Definition" position="194\6" endposition="197\32">
<Functor-Definition MMLId="XTUPLE_0:12">
 <Redefine occurs="false"/>
 <InfixFunctor–Pattern formatdes="O43[0(1)1]" formatnr="37" spelling="proj1"
  position="194\12" patternnr="21" absolutepatternMMLId="XTUPLE_0:12" constr="K18"
  absoluteconstrMMLId="XTUPLE_0:9" origconstrnr="0">
  <Loci/>
  <Loci>
   <Locus idnr="25" varidkind="Identifier" spelling="X" position="193\7" origin="Constant"
    kind="Constant" serialnr="54" varnr="1"/>
  </Loci>
 </InfixFunctor–Pattern>
 <Type–Specification>
  <Standard–Type nr="1" formatnr="3" patternnr="2" absolutepatternMMLId="HIDDEN:2"
   spelling="set" sort="Mode" constrnr="2" absoluteconstrMMLId="HIDDEN:2" originalnr="0"
   position="198\21">
   <Arguments/>
  </Standard–Type>
 </Type–Specification>
 <Definiens>...</Definiens>
</Block>
```

Then, if we consider a relation, then it is more natural to call this projection the domain of the relation. Hence, as in the article `RELAT_1`, we can introduce a convenient synonym for this operation (for the same constructor) under the assumption that the argument is of the relation type:

Listing 3 Domain as a synonym for the first projection

```
notation :: RELAT_1
  let R be Relation;
  synonym dom R for proj1 R;
end;
```

Listing 4 ESM representation of the domain

```
<Block kind="Notation–Block" position="103\8" endposition="106\3">
<Item kind="Loci–Declaration" position="104\5" endposition="104\19">
<Loci–Declaration>
 <Qualified–Segments>
  <Explicitly–Qualified–Segment position="104\7">
   <Variables>
    <Variable idnr="29" spelling="R" position="104\7"
              kind="Constant" serialnr="31" varnr="1"/>
   </Variables>
   <Standard–Type nr="5" formatnr="44" patternnr="6" position="104\19"
    spelling="Relation" sort="Expandable–Type" absolutepatternMMLId="RELAT_1:1">
    <Arguments/>
   </Standard–Type>
  </Explicitly–Qualified–Segment>
 </Qualified–Segments>
</Loci–Declaration>
</Item>
<Item kind="Func–Synonym" position="105\13" endposition="105\27">
<Func–Synonym>
 <InfixFunctor–Pattern formatdes="O12[0(1)1]" formatnr="45" spelling="dom"
```

```
  origpatternnr="16" absoluteorigpatternMMLId="XTUPLE_0:12" patternnr="27"
  absolutepatternMMLId="RELAT_1:1" constr="K32"
  absoluteconstrMMLId="XTUPLE_0:9" origconstrnr="0" position="105\13">
  <Loci/>
  <Loci>
   <Locus idnr="29" varidkind="Identifier" spelling="R" position="104\7" origin="Constant"
    kind="Constant" serialnr="31" varnr="1"/>
  </Loci>
 </InfixFunctor-Pattern>
```

Finally, when the relation happens to be defined on a given set X, we may use this information as in the article `RELSET_1` to redefine the domain with a more specific result type, i.e. a subset of X (this creates a new constructor):

Listing 5 Redefinition of the domain

```
definition :: RELSET_1
  let X be set;
  let R be X-defined Relation;
  redefine func dom R -> Subset of X;
end;
```

Listing 6 Excerpt from ESM representation of the redefinition

```
<Functor-Definition>
 <Redefine occurs="true"/>
 <InfixFunctor-Pattern formatdes="O2[0(1)1]" formatnr="45" spelling="dom" position="113\18"
  origpatternnr="23" absoluteorigpatternMMLId="RELAT_1:1" patternnr="35"
  absolutepatternMMLId="RELSET_1:1" constr="K48"
  absoluteconstrMMLId="RELSET_1:1" origconstrnr="42"
  absoluteorigconstrMMLId="XTUPLE_0:9">
  <Loci/>
  <Loci>
   <Locus idnr="22" varidkind="Identifier" spelling="R" position="112\23" origin="Constant"
    kind="Constant" serialnr="40" varnr="2"/>
  </Loci>
 </InfixFunctor-Pattern>
 <Type-Specification>
  <Standard-Type nr="5" formatnr="34" patternnr="4" position="113\30"
   spelling="Subset" sort="Expandable-Type" absolutepatternMMLId="SUBSET_1:2">
   <Arguments>
    <Simple-Term idnr="11" spelling="X" position="113\35" origin="ReservedVar"
     sort="Constant" serialnr="3" varnr="1"/>
   </Arguments>
  </Standard-Type>
 </Type-Specification>
```

Thanks to the absolute references provided within the ESM format it is possible to easily distinguish the two notions represented by different constructors (`absoluteconstrMMLId = "XTUPLE_0:9"` and `absoluteconstrMMLId = "RELSET_1:1"`) and three different patterns (`absolutepatternMMLId = "XTUPLE_0:12"`, `absolutepatternMMLId = "RELAT_1:1"` and `absolutepatternMMLId = "RELSET_1:1"`) used when applying this operation in various contexts.

2.3 Decoding Mizar definitions

All the available internal formats may still be used for specific tasks related to processing Mizar articles. However, in many cases the implementation may significantly benefit from utilizing the ESM format. The potential transition requires understanding both the formerly used data structures and the new ESM capabilities. For example, in order to demonstrate how the use of ESM facilitates the unique identification of all components of formulas, let us first analyze the formula `A c= A \/ B` using only its XML representation:

`.xml` file	`.aco` file
`<Pred kind="R" nr="3" pid="4">`	`<SignatureWithCounts>`
`<Var nr="1"/>`	`<ConstrCounts name="HIDDEN">`
`<Func kind="K" nr="6" pid="16">`	`<ConstrCount kind="M" nr="2"/>`
`<Var nr="1"/>`	`<ConstrCount kind="R" nr="2"/>`
`<Var nr="2"/>`	`</ConstrCounts>`
`</Func>`	`<ConstrCounts name="TARSKI">`
`</Pred>`	`<ConstrCount kind="M" nr="2"/>`
	`<ConstrCount kind="R" nr="5"/>`
	`<ConstrCount kind="K" nr="4"/>`
	`</ConstrCounts>`
	`<ConstrCounts name="XBOOLE_0">`
	`<ConstrCount kind="M" nr="2"/>`
	`<ConstrCount kind="V" nr="1"/>`
	`<ConstrCount kind="R" nr="8"/>`
	`<ConstrCount kind="K" nr="9"/>`
	`</ConstrCounts>`

To identify which inclusion and which union are used in the formula, the following steps should be done:

1. From values `kind="R"` and constructor number `nr="3"` and content of `.aco` file we can conclude that the inclusion is the first predicate in `TARSKI` article (numeral 3 is bigger than 2 in the line `<ConstrCount kind="R" nr="2"/>` and lower than 5 in the line `<ConstrCount kind="R" nr="5"/>`).
2. From values `kind="R"` and pattern number `pid="4"` and the line:
   ```
   <Pattern kind="R" nr="1" aid="TARSKI" formatnr="7"
     constrkind="R" constrnr="3" relnr="4">
   ```
   of `.eno` file (`pid="4"` = `relnr="4"`) we know the format number `formatnr="7"`.
3. From values `kind="R"` and `formatnr="7"` and the line:
   `<Format kind="R" nr="7" symbolnr="4" argnr="2" leftargnr="1"/>`
   of `.frx` file (`formatnr="7"` = `nr="7"`) we know the number of used symbol `symbolnr="4"`, and we know that the predicate has two arguments (`argnr="2"`) and arguments are placed as one on the left side of the symbol (`leftargnr="1"`) and one on the right side of the symbol.
4. From `kind="R"` and `symbolnr="4"` and the line:
   `<Symbol kind="R" nr="4" name="c="/>` of `.dcx` file (`symbolnr="4"` = `nr="4"`) we conclude that `c=` is used as a symbol of the relation (`name="c="`).
5. Similar reasoning can be done about the operation coded as `kind="K"` and `nr="6"` .

And when we look at the corresponding piece of the `.esx` file:

Table 3 XML elements of basic concepts.

Terms	Types	Formulas
Aggregate-Term	Clustered-Type	Biconditional-Formula
Circumfix-Term	ReservedDscr-Type	Conditional-Formula
Forgetful-Functor-Term	Standard-Type	Conjunctive-Formula
Fraenkel-Term	Struct-Type	Disjunctive-Formula
Global-Choice-Term		Existential-Quantifier-Formula
Infix-Term		FlexaryConjunctive-Formula
Internal-Selector-Term		FlexaryDisjunctive-Formula
Numeral-Term		Multi-Attributive-Formula
Placeholder-Term		Multi-Relation-Formula
Private-Functor-Term		Negated-Formula
Qualification-Term		Private-Predicate-Formula
Selector-Term		Qualifying-Formula
Simple-Fraenkel-Term		Relation-Formula
Simple-Term		RightSideOf-Relation-Formula
it-Term		Universal-Quantifier-Formula

Listing 7

```
<Relation−Formula nr="4" formatnr="7" patternnr="4" absolutepatternMMLId="TARSKI:1"
     leftargscount="1" spelling="c=" sort="Relation−Formula" constrnr="3"
     absoluteconstrMMLId="TARSKI:1" originalnr="0" position="11\4">
 <Arguments>
  <Simple−Term idnr="4" spelling="A" position="11\1" origin="ReservedVar"
       sort="BoundVar" serialnr="1" varnr="1"/>
 <Infix−Term nr="17" formatnr="37" patternnr="16" absolutepatternMMLId="XBOOLE_0:2"
       leftargscount="1" spelling="\/" sort="Functor−Term" constrnr="6"
       absoluteconstrMMLId="XBOOLE_0:2" originalnr="0" position="11\9">
   <Arguments>
    <Simple−Term idnr="4" spelling="A" position="11\6" origin="ReservedVar"
         sort="BoundVar" serialnr="1" varnr="1"/>
    <Simple−Term idnr="5" spelling="B" position="11\11" origin="ReservedVar"
         sort="BoundVar" serialnr="2" varnr="2"/>
   </Arguments>
  </Infix−Term>
 </Arguments>
</Relation−Formula>
```

we can see that all this information is accessible in one place and no extra computations are required to identify used notions.

3 Main ESM grammar items

In this section we show a selection of ESM grammar items and examples of corresponding Mizar code. Presented data types are specifically related to the extension with respect to earlier WSM and MSM representations. Table 3 lists the names of all possible basic formula, type and term kinds now shared by all strict Mizar formats.

All various Mizar term categories are presented in Table 4 with examples of their applicability. Four possible basic Mizar type variants are represented as elements listed in Table 5, while Table 6 presents available formula categories, respectively.

Table 4 XML elements of terms with examples.

Term	Description	Example
Aggregate-Term	tuple structure	`ZeroStr(#A,a#)`
Circumfix-Term	bracketed term	`[:A,B:]`
Forgetful-Functor-Term	sub-tuple structure	`the ZeroStr of Gr`
Fraenkel-Term	Fraenkel operator	`{n where n is Nat: n is odd}`
Global-Choice-Term	global-choice operator	`the set`
Infix-Term	standard term	`A \/ B`
Internal-Selector-Term	structure selector (inside its definition)	`the carrier`
Numeral-Term	numeral	1
Placeholder-Term	argument of private definitions	`$1`
Private-Functor-Term	private functor (`deffunc`)	`F(1)`
Qualification-Term	type-cast operator	`1 qua Element of REAL`
Selector-Term	structure selector (with argument)	`the carrier of Gr`
Simple-Fraenkel-Term	Fraenkel operator	`the set of all n where n is Nat`
Simple-Term	constant	`a`
it-Term	definiendum representation	`it`

Table 5 XML elements of types with examples.

Types	Description	Example
Clustered-Type	type with adjectives	`finite set`
ReservedDscr-Type	dependent type in reservations	`Element of V`
Standard-Type	standard type	`object`
Struct-Type	structural type	`1-sorted`

3.1 Structures

Comparing to the information previously stored in WSM, the description of defined structures is significantly extended. Let us consider the following structure definition as an example.

```
reserve S1,S2 for 1-sorted;

definition
  let S1;
  let S2 be 1-sorted;
  struct (ModuleStr over S1, RightModStr over S2) BiModStr over S1,S2
  (#
    carrier -> set,
    addF, multF -> BinOp of the carrier,
    ZeroF, OneF -> Element of the carrier,
    lmult -> Function of [:the carrier of S1, the carrier:], the carrier,
    rmult -> Function of [:the carrier, the carrier of S2:], the carrier
  #);
end;
```

The same definition with annotations from corresponding pieces of `.esx` file may look like this:

Table 6 XML elements of formulas with examples.

Formulas	Description	Example
Biconditional-Formula	equivalence	`iff`
Conditional-Formula	implication	`implies`
Conjunctive-Formula	conjunction	`&`
Disjunctive-Formula	disjunction	`or`
Existential-Quantifier-Formula	existentially quantified formula	`ex x st P[x]`
FlexaryConjunctive-Formula	flexary conjunctive	`& ... &`
FlexaryDisjunctive-Formula	flexary disjunction	`or ... or`
Multi-Attributive-Formula	attributive formula	`{} is empty`
Multi-Relation-Formula	chain formula	`A c= B c= C`
Negated-Formula	negation	`not`
Private-Predicate-Formula	private predicate (`defpred`)	`P[set,Nat]`
Qualifying-Formula	explicit type qualification	`1 is object`
Relation-Formula	standard formula	`1 <= 2`
RightSideOf-Relation-Formula	tail of chain formula	`c= C`
Universal-Quantifier-Formula	universally quantified formula	`for x holds P[x]`

```
definition
  let S1;
     :: <Loci-Declaration> / <Qualified-Segments> / <Implicitly-Qualified-Segment>
  let S2 be 1-sorted;
     :: <Loci-Declaration> / <Qualified-Segments> / <Explicitly-Qualified-Segment>
  struct
  (ModuleStr over S1, RightModStr over S2) :: <Structure-Definition> / <Ancestors>
  BiModStr :: <Structure-Pattern>
  over S1,S2 :: <Structure-Pattern> / <Loci>
  (#  :: <Field-Segments>
  carrier :: <Field-Segment> / <Selectors>
  ->
  set, :: <Field-Segment> / <Standard-Type>
  addF, multF :: <Field-Segment> / <Selectors>
  ->
  BinOp of the carrier, :: <Field-Segment> / <Standard-Type>
  ZeroF, OneF -> Element of the carrier,
  lmult -> Function of [:the carrier of S1, the carrier:], the carrier,
  rmult -> Function of [:the carrier, the carrier of S2:], the carrier
  #);
end;
```

Apart from the representation of visible syntactic elements, the XML data structure of ESM now contains the following extra elements enclosed within the `<Structure-Patterns-Rendering>` container.

1. `<AggregateFunctor-Pattern>` representing the patterns for tuple terms encoded as `<Aggregate-Term>`. Aggregates represent concrete full structures. For example rings of integers with addition and multiplication as ring operations.
2. `<ForgetfulFunctor-Pattern>` representing the patterns for substructure terms `<Forgetful-Functor-Term>`. Forgetful terms can represent full structures or substructures which are ancestors (direct or indirect) of the structure, for example `the 1-sorted of B` is an indirect and `the ModuleStr of B` is a direct ancestor of some `B` of type `BiModStr`.

3. `<Strict-Pattern>` defining a special adjective `strict`. For example a ring is not a strict group, because it contains more selectors than those of a group. `strict` is generated automatically when a definition of a structure occurs. Regular adjectives, like `empty`, `finite`, etc. must be defined within regular definitional blocks.
4. `<Selectors-List> / <SelectorFunctor-Pattern>` representing the patterns of terms of the category encoded as `<Selector-Term>` Selector terms represent one field of a given structure, for example `the carrier of B`.

4 Applications

The proposed representation format is intended to facilitate various applications based on Mizar formalizations. Its validity and completeness has been initially tested and demonstrated with a collection of HTML files generated from Mizar source files allowing precise browsing through the library and exploring semantic links between notions. Another natural application of the extended ESM representation is in the system used by the journal Formalized Mathematics (FM)[2].

4.1 Linked Mizar Mathematical Library

As Mizar articles in the MML library had been continuously revised while Formalized Mathematics published their state at the time of the publication, an electronic counterpart of FM, called Journal of Formalized Mathematics (JFM), was developed in 1995 with the intention of representing the updated MML. The project initially funded by the ONR Grant N00014-95-1-1336 *Automated hyper-linking in an electronic mathematical proof-check journal* [3] continued till 2004. A central part of the project was a collection of HTML files generated from Mizar source files which offered users intuitive browsing through the library and exploring the inter-linked notions. After introducing in 2004 the internal XML-based format (`.xml` files) representing the result of the Mizar verifier's analyser pass, J. Urban used it to re-implement the HTML[4] linking part of JFM using XSL and extending the original format with useful extra functionality, e.g. rendering of complete proofs. Since then, the technology was frequently used by various external systems based on the MML semantic connections and linking (c.f. [3, 19, 16, 1, 11, 13]).

It should be noted that Urban's HTML rendering of Mizar articles is enriched with various elements invisible in the content of the original Mizar texts, and generated during the verification, such as: definitional theorems (the internal representation of definitions in the form of equivalence theorems) or expanded attribute clusters (sets of attributes appearing in the text extended with automatically calculated consequences based on registrations available in the environment). The representation thus realized is therefore richer than the content of the article seen at the level of the Mizar text. However, generating HTML documents on the basis of `.xml` files involves the necessity of multiple "recalculations" of numerical representations of objects (formats, patterns or constructors). It also loses the original structure of logical formulas as a result of translating these formulas into semantic correlates needed by the checker or expanding local variables introduced with the `set` construct).

[2] ISSN 1426-2630 (Print), eISSN 1898-9934 (Online), `http://fm.mizar.org`
[3] `https://apps.dtic.mil/dtic/tr/fulltext/u2/a322951.pdf`
[4] Available on-line at `http://mizar.uwb.edu.pl/version/current/html/`

Table 7 Examples of link anchor names.

kind	name
attribute	`#articlename_V1_P1`
predicate	`#articlename_R1_P1`
functor	`#articlename_K1_P1`
expandable mode	`#articlename_ME1_P1`
regular mode	`#articlename_M1_P1`
selector	`#articlename_U1_P1`
aggregate	`#articlename_G1_P1`
structure	`#articlename_L1_P1`
theorem	`#ARTICLENAME:1`
definition	`#ARTICLENAME:def1`
scheme	`#ARTICLENAME:sch1`
local reference	`#Lab_S1_L1`
local predicate	`#PrP_S1_V1`
local functor	`#PrF_S1_V1`

The potential utility of our new ESM syntactic-semantic format (`.esx` files) can also be demonstrated by a similar collection of hyper-linked Mizar articles (see Supplementary Material). In comparison to the representation generated and partially reconstructed from the `.xml` files, it is now possible to render the semantic connections between linked notions and at the same preserve the original syntactic structure of the underlying Mizar text.

The ready-available information was used to implement a system of links (anchor names) for the definitions of:

- all basic concepts, i.e. predicates, adjectives, types, operations and structures
- local predicates and local functors,
- local labels and references to external statements, definitions, and schemas.

The links from concepts to their definitions consist of: the ID of the article in which the term was defined, concept type (`V, R, M, K, O, L, G, U`), constructor number of a given concept and the pattern number of a given concept preceded by the letter `P`. Table 7 presents the details of the link naming convention used.

As an example, let us look at the representation of the adjective `odd` defined as the antonym of `even` in the article `abian.miz`. `odd` is represented as `abian_V1_P2`. We can see the difference between the constructor number (1) and the pattern number (2). It results from the fact that antonyms generate a new pattern, and do not generate a new constructor (antonyms inherit the constructor numbers of the originals).

Links to local predicates consist of the `PrP` prefix, a serial number preceded by `S` and the current predicate number available in the given reasoning block preceded by `V`. Links to local functors have the same structure but with the `PrF` prefix. Finally, links to local labels use the prefix `Lab`.

References to theorems, definitions, and schemes consist of the article identifier from which the item is derived and the corresponding number preceded by `def` for a definition or `sch` for a scheme.

Considering the redefinition of a relation's domain described in Section 2.1, we may see its rendering at `http://alioth.uwb.edu.pl/~artur/mmlesx/relset_1.html#relset_1_K1_P1` with a link to the synonymous notion being redefined at `http://alioth.uwb.`

edu.pl/~artur/mmlesx/relat_1.html#xtuple_0_K9_P1. The internal XML-based HTML representation at http://mizar.uwb.edu.pl/version/current/html/relset_1.html#K1 lacks this syntactic information and offers only a link to the original constructor (http://mizar.uwb.edu.pl/version/current/html/xtuple_0.html#K9).

The system of links can be further extended e.g. with references to the origin of constants, quantified variables, scheme variables etc. Please note that although the aim of the project was to reproduce the text of the Mizar articles as faithfully as possible, the differences in the use of spaces, line breaks or brackets are currently disregarded. Accurate reproduction of these aspects would require more purely textual information generated by the Mizar parser to be recorded and stored in `.wsx` files.

4.2 Formalized Mathematics

Formalized Mathematics (FM) publishes papers based on regular Mizar formalizations accepted for inclusion into the Mizar Mathematical Library following a round of human peer-review. After acceptance, the underlying Mizar scripts are automatically translated into a LaTeX format [2] and the resulting generated natural language (English) texts become available as traditional mathematical papers downloadable as PDF files. The current implementation the of software responsible for the translation is based on a number of `XSLT` style-sheets which convert `.wsx` files representing parse-trees of given Mizar articles into a series of XML files containing human readable meta-text with increasing level of semantic detail [6]. As the original Mizar scripts are plain text files encoded with standard ASCII, traditional mathematical symbols like $\cup$, Σ, or $\int$ cannot be directly used in Mizar texts. To render them in the LaTeX documents, authors of Mizar formalizations can propose their preferred *translation patterns* to FM editors. These patterns allow changing formal and often technical-looking Mizar statements into more natural representation resembling standard mathematical notation using traditional symbols or a fixed placement and order of arguments. Sometimes these patterns become more complex, e.g. in the case of matrices, where the translation needs a special language construct rather than a simple symbolic replacement.

The current FM translation method based primarily on the `.wsx` files has the downside that some different notions introduced in the Mizar script are indistinguishable without special processing by more advanced modules of the Mizar verifier. For example, the multiplication of complex numbers and the multiplication of elements of a ring, when they both are written as infix operations utilizing the same symbol. Such shortcomings are overcome by the richer information present in the new proposed ESM format. The corresponding new `.esx` representations look like this:

Listing 8

```
<Functor-Definition MMLId="E1:1">
 <InfixFunctor-Pattern formatdes="O43[1(2)1]" formatnr="80" spelling="*"
    position="24\9" patternnr="224" absolutepatternMMLId="E1:2" constr="K465"
    absoluteconstrMMLId="E1:1" origconstrnr="0">
```

Listing 9

```
<Functor-Definition MMLId="E1:2">
 <InfixFunctor-Pattern formatdes="O43[1(2)1]" formatnr="80" spelling="*"
    position="33\9" patternnr="225" absolutepatternMMLId="E1:3" constr="K466"
    absoluteconstrMMLId="E1:2" origconstrnr="0">
```

We can observe that both operations have the same `formatnr="80"`, but they can now be uniquely identified by values of the `patternnr` and `constr` attributes. Consequently, the future FM representations of Mizar articles can be improved accordingly once the translation is based on patterns rather than formats.

5 Conclusions

In this paper we announced the existence of an extended XML-based data format simplifying the access to mathematical notions formalized in Mizar and available as part of the Mizar Mathematical Library, called Even more Strict Mizar (ESM). It is designed to combine and provide an easy access to both syntactic and semantic data of the underlying Mizar scripts. The extra information should allow creating various applications of the Mizar library requiring fullest possible information to be retrieved from the formalization files, especially using external general-purpose XML processing libraries (e.g. `dom4j`[5] or RapidXml[6]). Additionally, the work on the new language helped to analyze and improve the structure of already existing Mizar XML file formats (WSM and MSM).

References

1 Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning*, 52(2):191–213, 2014. `doi:10.1007/s10817-013-9286-5`.

2 Grzegorz Bancerek. Automatic translation in Formalized Mathematics. *Mechanized Mathematics and Its Applications*, 5(2):19–31, December 2006.

3 Grzegorz Bancerek. Information retrieval and rendering with MML query. In Jonathan Borwein and William Farmer, editors, *Mathematical Knowledge Management*, volume 4108 of *Lecture Notes in Computer Science*, pages 266–279. Springer Berlin Heidelberg, 2006. `doi:10.1007/11812289_21`.

4 Grzegorz Bancerek, Czesław Byliński, Adam Grabowski, Artur Korniłowicz, Roman Matuszewski, Adam Naumowicz, and Karol Pąk. The role of the Mizar Mathematical Library for interactive proof development in Mizar. *Journal of Automated Reasoning*, 61(1):9–32, June 2018. `doi:10.1007/s10817-017-9440-6`.

5 Grzegorz Bancerek, Czesław Byliński, Adam Grabowski, Artur Korniłowicz, Roman Matuszewski, Adam Naumowicz, Karol Pąk, and Josef Urban. Mizar: State-of-the-art and beyond. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Intelligent Computer Mathematics – International Conference, CICM 2015, Washington, DC, USA, July 13–17, 2015, Proceedings*, volume 9150 of *Lecture Notes in Computer Science*, pages 261–279. Springer, 2015. `doi:10.1007/978-3-319-20615-8_17`.

6 Grzegorz Bancerek, Adam Naumowicz, and Josef Urban. System description: XSL-based translator of Mizar to LaTeX. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *Intelligent Computer Mathematics – 11th International Conference, CICM 2018, Hagenberg, Austria, August 13–17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 2018. `doi:10.1007/978-3-319-96812-4_1`.

7 Czesław Byliński and Jesse Alama. New developments in parsing Mizar. In Johan Jeuring, John A. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge, editors, *Intelligent Computer Mathematics 11th International Conference, AISC 2012, 19th Symposium, Calculemus 2012, 5th International Workshop, DML 2012, 11th*

[5] `https://dom4j.github.io/`
[6] `http://rapidxml.sourceforge.net/`

International Conference, MKM 2012, Systems and Projects, volume 7362 of *Lecture Notes in Artificial Intelligence*, pages 427–431. Springer-Verlag, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-31374-5_30.

8 Ingo Dahn. Interpretation of a Mizar-like logic in first-order logic. In Ricardo Caferra and Gernot Salzer, editors, *FTP (LNCS Selection)*, volume 1761 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 1998. doi:10.1007/3-540-46508-1_9.

9 Mihnea Iancu, Michael Kohlhase, Florian Rabe, and Josef Urban. The Mizar Mathematical Library in OMDoc: Translation and applications. *Journal of Automated Reasoning*, 50(2):191–202, February 2013. doi:10.1007/s10817-012-9271-4.

10 Cezary Kaliszyk and Karol Pąk. Isabelle import infrastructure for the Mizar Mathematical Library. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *Intelligent Computer Mathematics – 11th International Conference, CICM 2018, Hagenberg, Austria, August 13–17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2018. doi:10.1007/978-3-319-96812-4_13.

11 Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *CoRR*, abs/1310.2805, 2013. URL: http://arxiv.org/abs/1310.2805.

12 Roman Matuszewski and Piotr Rudnicki. Mizar: The first 30 years. *Mechanized Mathematics and Its Applications, Special Issue on 30 Years of Mizar*, 4(1):3–24, March 2005.

13 Kazuhisa Nakasho. Development of a flexible Mizar tokenizer and parser for information retrieval system. In Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki, editors, *Proceedings of the 2019 Federated Conference on Computer Science and Information Systems, FedCSIS 2019, Leipzig, Germany, September 1–4, 2019*, volume 18 of *Annals of Computer Science and Information Systems*, pages 77–80, 2019. doi:10.15439/2019F151.

14 Adam Naumowicz and Radosław Piliszek. Accessing the Mizar library with a weakly strict Mizar parser. In Michael Kohlhase, Moa Johansson, Bruce R. Miller, Leonardo de Moura, and Frank Wm. Tompa, editors, *Intelligent Computer Mathematics – 9th International Conference, CICM 2016, Bialystok, Poland, July 25–29, 2016, Proceedings*, volume 9791 of *Lecture Notes in Computer Science*, pages 77–82. Springer, 2016. doi:10.1007/978-3-319-42547-4_6.

15 Karol Pąk. Combining the Syntactic and Semantic Representations of Mizar Proofs. In Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki, editors, *Proceedings of the 2018 Federated Conference on Computer Science and Information Systems, FedCSIS 2018, Poznan, Poland, September 9–12, 2018*, volume 15 of *Annals of Computer Science and Information Systems*, pages 145–153. IEEE, 2018. doi:10.15439/2018F248.

16 J. Urban, G. Sutcliffe, S. Trac, and Y. Puzis. Combining Mizar and TPTP Semantic Presentation and Verification Tools. *Studies in Logic, Grammar and Rhetoric*, 18(31):121–136, 2009.

17 Josef Urban. XML-izing Mizar: Making semantic processing and presentation of MML easy. In Michael Kohlhase, editor, *Mathematical Knowledge Management, 4th International Conference, MKM 2005, Bremen, Germany, July 15–17, 2005, Revised Selected Papers*, volume 3863 of *Lecture Notes in Computer Science*, pages 346–360. Springer, 2005. doi:10.1007/11618027_23.

18 Josef Urban. MizarMode – an integrated proof assistance tool for the Mizar way of formalizing mathematics. *Journal of Applied Logic*, 4(4):414–427, 2006.

19 Josef Urban. MoMM – fast interreduction and retrieval in large libraries of formalized mathematics. *Int. J. on Artificial Intelligence Tools*, 15(1):109–130, 2006. URL: http://ktiml.mff.cuni.cz/~urban/MoMM/momm.ps.

20 Josef Urban, Piotr Rudnicki, and Geoff Sutcliffe. ATP and presentation service for Mizar formalizations. *Journal of Automated Reasoning*, 50(2):229–241, February 2013. doi:10.1007/s10817-012-9269-y.