Expressing the Notion of a Mathematical Structure in the Formal Language of Mizar

Adam Grabowski(✉)

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

Abstract. In the paper, the problem with the translation of ordinary mathematical structures from the natural language into the formal one is identified. As a concrete example of the system I used the Mizar system, the interactive proof assistant for the formalization of mathematics. Chosen testbed, both for new developments, and for revising old ones, was the theory of $\mathbb{Z}$-modules.

Keywords: Formal language · Mathematical knowledge management · Mizar

1 Introduction

Growing popularity of computerized mathematical proof-assistants (Voevodsky who won the Fields medal in 2002 underlines the future of computer approach building new foundations of mathematics—univalent foundations) raises a number of new problems which should be solved in order to meet expectations of researchers. It is important, that the formal approach should be flexible enough both to be easily writeable and understandable for human and to allow for further generalizations. In recent years, traditional model of printed contribution fixed for years could be adjusted to take into account the possibilities given by contemporary media where such knowledge is stored.

In this paper we focus on the area where mathematical structures can be extensively used and their formal counterpart can be tuned accordingly as it was already formalized within machine-verified mathematical knowledge repository. We mean the theory of $\mathbb{Z}$-modules certified with the help of the Mizar system [9]. The problem was, how to translate these objects expressed in the natural language used by mathematicians into the formal language of Mizar. The topic is quite well represented in the Mizar Mathematical Library [1], and looks promising for the mathematics as a whole—modules are a basic apparatus for representation theory, the tool for representing many other areas of mathematics.

2 The Mizar System

The main aim of the Mizar system—the project steered by Andrzej Trybulec from early seventies of the previous century – was to develop a formal approach to mathematics which allows for faithful encoding of the definitions and

A. Gruca et al. (eds.), *Man-Machine Interactions 5*,
Advances in Intelligent Systems and Computing 659, DOI 10.1007/978-3-319-67792-7_26

theorems written in natural language in order to be verified for correctness by computers. This formal approach should be flexible enough to be understandable for ordinary human without much pain, so one of the very basic points was to be as close as possible to mathematical vernacular. On the other hand one should have in mind the strictness and the relative simplicity of the grammar of the artificial language in order to be easily scanned by the parser of the Mizar system.

The second ingredient of the system is the repository of formal texts. The Mizar Mathematical Library (MML) [9] is based on Tarski-Grothendieck set theory, which is very close to the one used by the majority of mathematicians [17]. Hence it is not very strange that general topology is one of the widely represented parts of mathematics within this repository of knowledge. Among the large formalization projects of the Mizar community, two were connected with topology. The first one was the formalization of Jordan curve theorem, resulting in many articles written in tight cooperation with Japanese Mizar group. The second one, the formalization of *A Compendium of Continuous Lattices* by Gierz et al. [19] (see [9]), although originally meant to be placed within lattice theory, eventually was driven into the direction of category theory and topology.

Original motivation for our paper was to describe some of the issues raised in the process of formalizing important mathematical structures – modules (or vector spaces). Mizar structures have shown its usefulness when modelling the theory of tolerance approximation spaces connected with rough set theory defined by Pawlak [5,20]. But soon we realized that in order to do this properly (at least to use as much expressive power of the Mizar language as we can), we should lift two fundamental notions into the common ground—of the descendant of topological spaces merged with approximation spaces. We have observed that developing alternative background for already well-established area of formalized knowledge can cause many troubles. $\mathbb{Z}$-modules as our target occurred rather accidentally, but vector spaces are one of the most frequently used mathematical objects. Of course, the hierarchy of algebraic structures is much richer than that of topological spaces.

3 Towards Algebraic Hierarchy

The algebraic hierarchy is crucial to automate mathematical knowledge by means of automated proof-assistants: in **Nuprl** it was developed to support computational abstract algebra. In **Coq** more than one algebraic hierarchy exists, we name two of them: One was constructed as part of the FTA project to prove the fundamental theorem of algebra, another one was used in the formalization of the Feit-Thompson Theorem [4]. In the **HOL/Isabelle** Archive of Formal Proofs one finds a number of proof libraries devoted to algebraic domains. Lately in **ACL2** an algebraic hierarchy has been built in order to support reasoning about Common Lisp programs [12].

All algebraic structures in Mizar are defined in similar manner: first we have to give a structure, where names of fields (called selectors) with their specification (the type and the arity) are given. In our concrete case there were

```
definition
  struct (1-sorted) addMagma (# carrier -> set,
                                  addF -> BinOp of the carrier #);
end;
```

and

```
definition
  struct (ZeroStr,addMagma) addLoopStr (# carrier -> set,
                              addF -> BinOp of the carrier,
                            ZeroF -> Element of the carrier #);
end;
```

Structures in Mizar can be used to model mathematical notions like groups, topological spaces, categories, etc. which are usually represented as tuples. A structure definition contains, therefore, a list of *selectors* to denote its fields, characterized by their name and type, e.g.:

```
definition
  struct multMagma (# carrier -> set,
                        multF -> BinOp of the carrier #);
end;
```

where `multMagma` is the name of a structure with two selectors: an arbitrary set called its `carrier` and a binary operation on it, called `multF`. This structure can be used to define a group, but also upper and lower semilattices [7], so in fact any notion that is based on a set and a binary operation on it. It should be observed that the above structure does not define a group yet (nor any other more concrete object), because there is no information on the properties of `multF`. The structure is just a basis for developing a theory. In practice, after introducing a required structure, a series of attributes is also defined to describe the properties of certain fields.

As mentioned before, the above `multMagma` structure can be used to define notions which are not only groups. Still, the operation in such structures inherit the name `multF`, because the current Mizar implementation does not provide a mechanism to introduce synonyms for selectors (or whole structures). Therefore, in cases when a different name is frequently used in standard mathematical practice, it may be better to introduce a different structure. For example, lattice operations are commonly called `meet` and `join`, so a lower semilattice may be better encoded as:

```
definition
  struct /\-SemiLattStr (# carrier -> set,
                            L_meet -> BinOp of the carrier #);
end;
```

Mizar supports multiple inheritance of structures that makes a whole hierarchy of interrelated structures available in the Mizar library, with the `1-sorted`

structure being the common ancestor of almost all other structures. For example, formalizing topological groups in Mizar can be done by independently defining and developing group theory and the theory of topological spaces, and then merging these two theories together [6] based on a new structure, e.g.:

```
definition
  struct (1-sorted) TopStruct (# carrier -> set,
     topology -> Subset-Family of the carrier #);
end;

definition
  struct (multMagma, TopStruct) TopGrStr (# carrier -> set,
             multF -> BinOp of the carrier,
          topology -> Subset-Family of the carrier #);
end;
```

The advantage of this approach is that all notions and facts concerning groups and topological spaces are naturally applicable to topological groups. Let us note that when introducing a new structure, the inherited selectors can be listed in any order, as far as relations between them are preserved. The list of names of ancestor structures is put in brackets before the name of the structure being defined. Concrete mathematical objects, e.g. the additive group of integers are introduced with *aggregates*—special term constructors defined automatically by the definition of a structure, e.g.: `multMagma(#INT,addint#)`, where `INT` is the set of integers, and `addint` represents the addition function. It is necessary that all terms used in the aggregate have the respective types declared in the structure's definition. In our example `INT` is obviously a set, and `addint` must be of type `BinOp of INT`. Every structure defines implicitly a special attribute, **strict**. The corresponding adjective's meaning is that an object of a structure type contains nothing more, but the fields defined for that structure. For example, a term with structural type based on `TopGrStr` may be **strict** `TopGrStr`, but it is neither **strict** `multMagma`, nor **strict** `TopStruct`. Clearly, every term constructed using a structure's aggregate is **strict**.

Finally, the Mizar language has means to restrict a given term with a complex structure type to its well-defined subtype. This special term constructor, the *forgetful functor* also utilizes the structure's name, e.g. `the multMagma of G`, where `G` has a potentially wider type which inherits the `multMagma` structure. Again, such terms are **strict**, with respect to the given structure type.

Magmas' and loops' names are credited to Bourbakists' unifying approach to algebra. Then comes the structure of left module over a ring (actually 1-sorted, which is sometimes called *setoid* in the world of proof-assistants) (Fig. 1).

```
definition let F be 1-sorted;
  struct (addLoopStr) VectSpStr over F (# carrier -> set,
     addF -> BinOp of the carrier,
    ZeroF -> Element of the carrier,
    lmult -> Function of [:the carrier of F,the carrier:], the carrier #);
end;
```

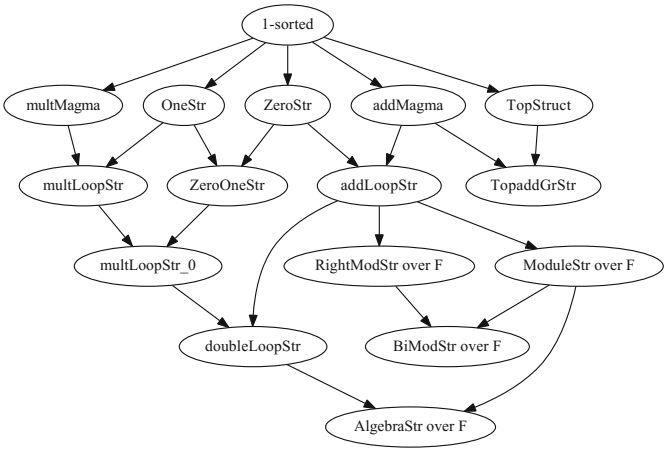

Fig. 1. Net of basic algebraic structures in the Mizar Mathematical Library [10]

where `[:A,B:]` stands for the Cartesian product of sets A and B. What was confusing, dual structures with right multiplication were named more traditionally—`RightModStr`. We decided to unify the approach and to better reflect algebraic convention, eventually changed the name into `ModuleStr over F`. But there is also another structure informally meaning the same:

```
definition
  struct (addLoopStr) RLSStruct (# carrier -> set,
             ZeroF -> Element of the carrier,
              addF -> BinOp of the carrier,
              Mult -> Function of [:REAL, the carrier:], the carrier #);
end;
```

Essentially, the difference between structures of the form

$$\langle L, 0, +, \cdot \rangle \qquad \text{vs.} \qquad \langle L, +, 0, * \rangle$$

(besides the ordering of fields which is obviously not very binding) is in the result type of both multiplications: $*$ is a function, where $\mathbb{R}$ appears in the domain, in the earlier case the set of all real numbers is replaced by the arbitrary set, and also the name space is different.

4 Attributes

The (part of) hierarchy of algebraic structures deliver only a signature for corresponding algebras; the real semantics is given by axioms. In Mizar formalism, axioms are defined as adjectives (called also attributes). We give some basic ones below, leaving them without much comment as the names and the syntax is explanatory enough—at least we hope so.

```
definition let IT be addMagma;
  attr IT is Abelian means              :: RLVECT_1:def 2
  for v,w being Element of IT holds v + w = w + v;
  attr IT is add-associative means      :: RLVECT_1:def 3
  for u,v,w being Element of IT holds (u + v) + w = u + (v + w);
end;

definition
  let F be add-associative right_zeroed right_complementable Abelian
    associative well-unital distributive non empty doubleLoopStr;
  mode LeftMod of F is scalar-distributive vector-distributive
    scalar-associative scalar-unital add-associative right_zeroed
    right_complementable Abelian non empty VectSpStr over F;
end;
```

The above definition looks too technical from human point of view. Remember however, that this plays a role of a macro expanding every time when used [1], and for a human it is enough to write just `LeftMod of F` with underlying F which is a ring; the rest is provided by Mizar typing mechanisms (understanding all the properties, catching the hierarchy of notions, etc.).

The original definition of $\mathbb{Z}$-module in [2]—the first in the series formalizing these structures – was the following:

```
definition
  struct (addLoopStr) Z_ModuleStruct (# carrier -> set,
     ZeroF -> Element of the carrier,
      addF -> BinOp of the carrier,
      Mult -> Function of [:INT, the carrier:], the carrier #);
end;
```

and this structure was equipped with the adjectives:

```
definition
  mode Z_Module is Abelian add-associative right_zeroed
    right_complementable scalar-distributive vector-distributive
    scalar-associative scalar-unital non empty Z_ModuleStruct;
end;
```

Four attributes were newly defined with the isomorphic meaning as in the general case (we mean scalar- and vector-distributivity, scalar-associativity and the existence of scalar unit). There is no surprise that it is just a copy of `RLSStruct` with `REAL` exchanged by `INT`—the set of all real numbers has been replaced by the set of all integers. Hence many articles could be copied without much thinking (although, to be honest, there are some module-specific properties formalized, so our revision couldn't be of the form of deleting all the content).

Of course, vector spaces are just (left) modules over fields instead of rings. As vector spaces were main needed formalized objects, the authors of the original approach did not pay much attention to possible generalizations. Luckily enough, thank to the mechanism of identification of various (but isomorphic in different

theories) operations [18], most of the symbols remain untouched. For example, dealing with the field of real numbers, we have in the following registration contained in VECTSP_1:

```
registration let a,b be Element of F_Real, x,y be Real;
  identify a+b with x+y when a = x, b = y;
end;
```

So the symbols of real addition and the addition in the field of reals are identical (and unified).

5 The Revision

I decided to make a revision of the MML preventing from repetition of the existing knowledge and in the same time generalizing and unifying approaches to modules and vector spaces treated formally. The gains were obvious, and the work looked rather promising and not very time-consuming—at least our preliminary change of ZMODUL01 was rather easy (Table 1).

Table 1. Revision results

MML identifier	kB	LOC	kB after	LOC after
ZMODUL01	120.442	4136	64.983	2096
ZMODUL02	115.489	3863	55.448	1898
ZMODUL03	92.101	2602	93.689	2626
ZMODUL04	115.419	2987	117.879	3030
ZMODUL05	100.220	2986	107.969	3135
ZMODUL06	135.526	3568	138.144	3610
ZMODUL07	120.020	3430	115.496	3200
ZMATRLIN	207.060	5700	189.223	5488
GAUSSINT	81.166	2616	81.916	2625

Although there are tools for automatic replacement of references and symbols (among other mechanisms for automatic discovery of knowledge [5]), I also expected some changes which should be done by hand and this was something I did eventually. However I met some unexpected traps: matrices were defined on fields, and in the framework of the set of reals all desired properties were satisfied. As the ring of integers INT.Ring is obviously not a field, I had the choice either to drop my approach, or to generalize more knowledge than I was initially concerned with. Of course, this forced me to touch even such articles as MATRLIN or LAPLACE on matrices of linear maps or Laplace expansions. Eventually I ended up with integral domains, and as I noticed, for MML that was quite a good move (sooner or later one can face the problem of generalizing the current notion of determinant: it is not simple as dropping some more attributes as

determinants can be defined on matrices over noncommutative rings—Gelfand's quasideterminants are well-known counterparts to determinants in noncommutative algebras, but here the notion of a determinant is even not unique). Dealing with determinants of matrices of real numbers is rather safe, but note that in the light of our discussion of the formal distinction between modules with `REAL` and the carrier of the real space they are just two different (although isomorphic in some sense) approaches.

Note that all generalizations should be done from bottom to top – any superfluous assumptions in fundamental articles are inherited by all its descendants. As the final result, $\mathbb{Z}$-modules are defined in a very simple and intuitive way: `mode Z_Module is LeftMod of INT.Ring`, which fully agrees with the standard definition.

6 Some Statistics

The structure `RLSStruct` was originally used more often—2467 times in MML according to MML Query browsing tool (together with `CLSStruct` which is a version of the vector space in case of complex numbers). Analogous query in case of `VectSpStr`, which is more general, returned 1953 occurrences, so the idea of the choice of the approach based only on statistical data (follow the path which is used by more authors) can be misleading. Such similarities can be discovered automatically and the Library Committee of the Association of Mizar Users (of which the author is the head) suggests to remove unnecessary repetitions. The problem is that some proofs can be written by the copy-and-paste technique, which of course can be also discovered, but this is also the case of natural analogies between proofs [16], which are very frequent in mathematics.

One of the structures, `Z_ModuleStruct` was completely eliminated, as well as attributes which have been defined as axioms. As the MML is quite large, the revision can be measured by purely quantitative means. The change of the length is striking on two first articles in the series (by some 50% each)—this was the cost of knowledge repetitions. Then the number of lines of code (LOC) stabilizes, and sometimes it is even a little bit bigger (as a result of exchanging original arithmetical computations, which were quite efficient, with similar symbolic algebraic calculations, but not built-in in such an extent). The MML version with new generalized approach is 5.30.1229 (and six new articles were written until now in this new approach). The changes can be browsed in detail even by ordinary `diff` tool on Mizar distributions. At the current stage the revision is frozen, as new articles like [3] devoted to the topic use this approach.

7 Conclusion and Future Work

In the paper I tried to show how theoretically straightforward examples can lead to difficult problems during their translation from informal presentation in natural human language into formalism of the Mizar language, a variant of mathematical vernacular. Based on the example of $\mathbb{Z}$-modules we could observe

that even if the approach is given in a not satisfactory way, it can be corrected in a process of the so-called revision [11]. The part of the work could be less painful—the splitting of the original definition as we proposed and automatic replacement of the references into new ones. The level of generality is obviously higher in our approach, so we hope to open some new paths in the formalization of algebraic structures, especially in more abstract form.

The second part, which could be done gradually and with the possible use of automatic tools, is that this proposed new version should be consumed in the MML—the theorems and definitions which can be formalized in the more general way, should be formulated so. This would also enable reusing purely algebraic constructions in another areas of mathematics – as representation theory is quite important field; this will not be restricted for the Mizar library only, as the translation from the Mizar formalism into other formal languages are available [13]. Also external tools can be used in a limited way [15]. The Mizar language itself is also flexible enough to offer new mechanisms, as e.g. defining the so-called flexary connectives [14]. In the informal form of a mathematical publication written by human in natural language, such process could (and eventually led in real life, as it was in the world of rough sets [8]) to the sequence of papers generalizing the approach gradually. Hence it is also a kind of a problem for repository storing the knowledge. In the case of MML, some automatic enhancements are possible. The repetitions are deleted, shrinking the files, and opening the way to improve the overall algebraic framework available in the MML. But the work in the described case of $\mathbb{Z}$-modules is by no means finished, as still there are many articles using `RLSStruct`. I plan to revise all the approach in the nearest future—recent experiments have shown it should not fail.

Although natural language is rather flexible, formal counterpart benefits from the relative coherence of the existing approaches. Furthermore, on the contrary to the provers like EQP, Vampire or Prover9, we should stress the role of the interaction between human and the machine. The distinction between other interactive proof-assistants like HOL-Light or Isabelle is that the interaction is still important after the inclusion of human submission into the repository of texts: I have shown how the process of revisions can significantly improve the quality of a text, which can be automatically checked by the software for correctness and is readable for human in the same time.

References

1. Bancerek, G., Byliński, C., Grabowski, A., Korniłowicz, A., Matuszewski, R., Naumowicz, A., Pąk, K., Urban, J.: Mizar: state-of-the-art and beyond. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) CICM 2015, vol. 9150, pp. 261–279. Springer, Washington, DC (2015). doi:10.1007/978-3-319-20615-8_17
2. Futa, Y., Okazaki, H., Shidama, Y.: $\mathbb{Z}$-modules. Formaliz. Math. **20**(1), 47–59 (2012). doi:10.2478/v10037-012-0007-z
3. Futa, Y., Shidama, Y.: Lattice of $\mathbb{Z}$-module. Formaliz. Math. **24**(1), 49–68 (2016). doi:10.1515/forma-2016-0005

4. Gonthier, G., et al.: A machine-checked proof of the odd order theorem. In: ITP 2013, pp. 163–179. Springer, Rennes (2013). doi:10.1007/978-3-642-39634-2_14
5. Grabowski, A.: Automated discovery of properties of rough sets. Fundam. Inf. **128**(1–2), 65–79 (2013). doi:10.3233/FI-2013-933
6. Grabowski, A.: Efficient rough set theory merging. Fundam. Inf. **135**(4), 371–385 (2014). doi:10.3233/FI-2014-1129
7. Grabowski, A.: Mechanizing complemented lattices within Mizar type system. J. Autom. Reason. **55**(3), 211–221 (2015). doi:10.1007/s10817-015-9333-5
8. Grabowski, A.: Lattice theory for rough sets - a case study with Mizar. Fundam. Inf. **147**(2–3), 219–236 (2016). doi:10.3233/FI-2016-1406
9. Grabowski, A., Korniłowicz, A., Naumowicz, A.: Four decades of Mizar. J. Autom. Reason. **55**(3), 191–198 (2015). doi:10.1007/s10817-015-9345-1
10. Grabowski, A., Korniłowicz, A., Schwarzweller, C.: On algebraic hierarchies in mathematical repository of Mizar. In: Ganzha, M., et al. (eds.) FedCSIS 2016, vol. 8, pp. 363–371. IEEE, Gdansk (2016). doi:10.15439/2016F520
11. Grabowski, A., Schwarzweller, C.: Revisions as an essential tool to maintain mathematical repositories. In: MKM 2007, pp. 235–249. Springer, Hagenberg (2007). doi:10.1007/978-3-540-73086-6_20
12. Heras, J., Martín-Mateos, F.J., Pascual, V.: Modelling algebraic structures and morphisms in ACL2. Appl. Algebra Eng. Commun. Comput. **26**(3), 277–303 (2015). doi:10.1007/s00200-015-0252-9
13. 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). doi:10.1007/s10817-012-9271-4
14. Korniłowicz, A.: Flexary connectives in Mizar. Comput. Lang. Syst. Struct. **44**, 238–250 (2015). doi:10.1016/j.cl.2015.07.002
15. Naumowicz, A.: Automating Boolean set operations in Mizar proof checking with the aid of an external SAT solver. J. Autom. Reason. **55**(3), 285–294 (2015). doi:10.1007/s10817-015-9332-6
16. Pąk, K.: Improving legibility of formal proofs based on the close reference principle is NP-hard. J. Autom. Reason. **55**(3), 295–306 (2015). doi:10.1007/s10817-015-9337-1
17. Trybulec, A., Korniłowicz, A., Naumowicz, A., Kuperberg, K.: Formal mathematics for mathematicians. J. Autom. Reason. **50**(2), 119–121 (2013). doi:10.1007/s10817-012-9268-z
18. Urban, J., Rudnicki, P., Sutcliffe, G.: ATP and presentation service for Mizar formalizations. J. Autom. Reason. **50**(2), 229–241 (2013). doi:10.1007/s10817-012-9269-y
19. Gierz, G., et al.: A Compendium of Continuous Lattices. Springer, Heidelberg (1980)
20. Pawlak, Z.: Rough sets. Int. J. Parallel Program. **11**(5), 341–356 (1982)