Progress in the Independent Certification of Mizar Mathematical Library in Isabelle

Cezary Kaliszyk
Universität Innsbruck,
Technikerstr. 21a/2, 6020 Innsbruck, Austria
Email: cezary.kaliszyk@uibk.ac.at

Karol Pąk
University of Białystok,
Ciołkowskiego 1M, 15-245 Białystok, Poland
Email: pakkarol@uwb.edu.pl

***Abstract*—The Mizar Mathematical Library is one of the largest collections of machine understandable formal proofs encompassing many areas of today mathematics including results from algebra, analysis, topology, and lattice theory. The Mizar system has so far been the only tool able to completely process, certify, and make use of these developments. In this paper, we present the progress in the development of an independent certification mechanism of Mizar proofs based on the Isabelle logical framework. The approach allows rechecking the Mizar formal proofs based on a more succinct and more precisely specified formal infrastructure. Additionally, it necessitates a full formal specification of the mechanisms that ensure the correctness of the defined objects, in particular, the proofs that such mechanisms are correct. The development already covers an important part of the Mizar library foundations. We improve the mechanism for defining Mizar structures and show that it permits simpler validation of proof developments involving such objects. To demonstrate this, we perform a complete translation of the Mizar net of basic algebraic structures including their attributes and certify all the corresponding proofs in Isabelle.**

I. Introduction

Computer certified formal proofs are today one of the most important techniques used in formal methods. They are used to guarantee the correctness of compilers [1], operating systems [2], hardware [3], as well as to certify mathematical results that involve computation [4]. The Mizar system [5] is one of the oldest computer systems used to certify proofs. Its library, the Mizar Mathematical Library [6] (MML) contains today more than 1200 articles and 60000 proved theorems mainly about mathematics. The Mizar system has so far been the only tool able to process, fully certify, and make use of these formal proof developments.

Algebraic structures are one of the basic building blocks of formal proofs. They are crucial both for the foundations of mathematics and of computer science. This can be witnessed by the formal proof libraries of various interactive proof systems. Indeed, the standard library of Isabelle/HOL [7] defines more than three hundred type classes used in most of its Archive of Formal Proofs [8]. Coq uses its records which include properties in its algebraic foundations, both in the standard library [9], its constructive repository [10], and in the small scale reflection libraries [11] used as a foundation for the Four-Color theorem and the Odd-Order theorem proofs. Finally, the MML [6] includes more than a hundred structures which together with different attributes correspond to thousands of different algebraic structures. 74% of the Mizar articles depend directly or indirectly on algebraic structures, including the most important domains of mathematics developed in MML, such as topological spaces, vector spaces, lattices, and fuzzy sets [12].

Supported by Polish National Science Center grant decision n°DEC-2015/19/D/ST6/01473.

In this paper, we discuss the progress in our project attempting to certify the MML independently. We make use of the Isabelle logical framework [13] to specify the foundations of Mizar [14]. We further define a number of mechanisms that help to translate the Mizar definitions and proofs [15]. We investigate the set theoretic representation of algebraic structures and certify them in the Isabelle logical framework object logic corresponding to the Mizar foundations as well as translate a significant part of the Mizar algebraic structure foundations. After shortly introducing logical frameworks and Isabelle (Section II), as well as Mizar and the corresponding object logic (Section III), the particular contributions of this paper are:

- We provide an infrastructure for more elegant proofs of Mizar structure correctness conditions including structures with multiple fields (Section IV);
- We formalize all basic algebraic Mizar structures in Isabelle/Mizar together with their defining properties including structures that include other structures as components, such as a structure over a field and show that the defined Mizar structures are correctly handled in the presence of attributes and in particular that proofs about such defined algebraic structures can be concise and elegant (Section V).

II. Logical Frameworks and Isabelle

Nearly all interactive proof assistants today rely on one fixed logic. This allows optimizing a system for that foundations. However, a number of systems focused on modeling actual logical systems. These are referred to as *logical frameworks*, and later a number of such systems became useful not only for modeling the logic but also to work in the specified logic, referred to as *object logic*. The three major logical frameworks are Isabelle [7], Twelf [16], and MMT [17].

Isabelle is today one of the proof systems with the largest libraries of formally proved theorems. It is based on a simple type theory with a shallow polymorphism that is implemented in a manually checked kernel. The meta logic provides the user with a set of primitives that makes it convenient to define object logics. The most developed object logics are higher-order logic, untyped set theory, and Lamport's temporal logic of actions.

An Isabelle formalization consists of one or more **theory** files. A theory is a collection of definitions, proved theorems, and notations that allow nicer presentation of terms. An Isabelle **definition** introduces a new identifier that is equal or equivalent (equal as a boolean predicate) to a definition body. A **theorem** or **lemma** consists of the statement and the proof. For most of the proved theorems presented in the rest of the paper, we will omit the proofs, they are fully included in the development. Each **abbreviation** allows for convenient input or output syntax for more complicated terms, without introducing new definitions. These are useful if such a definition would always need to be unfolded and is nicer presented as folded to the user. Most Isabelle proofs are today written in the declarative Isar style [18]. There, intermediate statements are introduced using the **have** keyword and justified using proof methods. For the rest of the paper, the methods and tactics used for the justifications are not essential, it is important to note their correspondence to proof steps that are considered obvious for humans. Finally the **assume** keyword introduces assumptions in proof blocks and **show** is used to denote the goal that is local to the proof block that is to be checked by Isabelle.

III. Mizar and Corresponding Object Logic

Mizar is one of the pioneering systems for mathematics formalization that is widely-used and still under active development. The Mizar project from its beginning aimed to make a system for human readable formalization of mathematics, where:

- the proof style was designed to imitate style occurring in the informal mathematical practice,
- the type system tries to express how mathematicians use mathematical objects and how they categorize them.

Therefore, Mizar uses a rich type system and proof style, which makes formalization of mathematics more intuitive and human-readable than in other systems [19], where the main idea of proofs is easy to observe [20]. Such situation occurs especially if the author of a formal proof puts additional effort to manually improve readability or uses dedicated tools [21] that optimize the NP-complete problems of improving legibility [22]. Therefore, it is not surprising that the solutions used in Mizar have been an inspiration to implement the analogical solutions in other systems.

One of these pioneering works in this field was made by J. Harrison [23] who explored the Mizar language. The result of this work was the environment Mizar Mode for HOL enabling writing proofs in a Mizar declarative way [24]. The similar solutions were implemented in other procedural proof assistants, e.g., *Declare* [25], *Isar language for Isabelle* [18], *Mizar-light for HOL Light* [26], `miz3` *for HOL Light* [27], *MMode for Coq or declarative proof language (DPL) for Coq* [28]. However, the similarity between these environments and Mizar system generally is limited to a few rules that are similar to the rules of the S. Jaśkowski natural deduction style [29], responsible for the universal quantifier introduction, the thesis indication, the implication elimination, the introduction of the reasoning by cases. It is worth emphasizing that the way of justification of the reasoning steps in these environments is based on tactics of the particular system that are very different from Mizar `by` (its equivalent can be found only in Mizar Mode for HOL [23]).

Another significant advantage of the project Mizar, from the point of view of other formal systems, is the library of mathematical knowledge formalized in the Mizar system, MML. However, the exploration of these data requires the sophisticated language constructions and types of Mizar that do not have close equivalents in other systems.

The largest translation of Mizar has been done by Urban [30] to the TPTP first-order language. Although this translation has covered the important part of the MML, it does not constitute the accurate representation of the Fraenkel operator and scheme [29]. Additional work on this solution has enabled the creation of the extensive theorems database of Mizar Problems for Theorem Proving (MPTP) that is used in the process of comparing the performance of leading systems of automatic theorem proving, as well as during the machine learning of the MizAR proof advice system [31].

Kunčar [32] has attempted to recover the Mizar system in the type system of HOL Light. This approach has enabled the translation of the first few simpler theories as transparent higher-order logic theories, however it is not applicable to the whole MML where the more advanced features of the Mizar system type are used. Difficult to reconstruct, are Mizar type mechanisms that check whether some type is a subtype of another type, generate the type of term base on types of subterm, which eliminate inconsistent instantiations and in consequence speed up the verification process. Additionally, two equal terms in Mizar can possess two incompatible types (e.g., see reconsider [33]).

The statements of the theorems in the whole MML have been exported to the MMT logical framework [34]. This allows the use of various MMT services for MML, such as searching the library or providing proof advice, however does not include an independent verification of the proofs or proof automation.

Isabelle already has an object logic Isabelle/ZF [35] based on set theory. Already the foundational axioms of ZF differ from those of Tarski-Grothendieck, and the type system introduced by Mizar is very different from any of the existing object logics in Isabelle. Furthermore, the library of Isabelle/ZF and the automation provided is quite different from that of the proposed research.

We defined an object logic that provides Mizar-like foundations in [14]. Here, we briefly remind its construction. As the foundations of Mizar are based on Jaśkowski first-order natural deduction, we start with the Isabelle/FOL object logic. We introduce one meta-level type for Mizar sets and one for Mizar types. We introduce the constants that correspond to sets being of particular types and to combine types (the Mizar soft type system allows intersection types [36]), the indefinite description operator, as well as the axioms that specify these constants. With the Isabelle syntax mechanisms, we allow defining Mizar like syntax for statements and definitions, which can later be used to specify the Tarski-Grothendieck foundations of set theory and translate the first few articles of the MML.

IV. Structure Requirements

Formalizations of computer systems often need to refer to mathematical structures. In informal computer science practice, such proofs typically use ordered tuples for such structures. For example $\langle G, +, 0\rangle$ could be an additive group and $\langle G, \cdot, 1\rangle$ a multiplicative one. In the informal approach, the expression "the group $\langle G, +, 0\rangle$" provides two kinds of information simultaneously: a signature and its properties. The signature says that it is a structure containing the set G, a binary operation + and a given element of the set 0. The properties are given as three group axioms. A formal approach to reason about such structures taken by the Mizar system attempts to avoid independent definitions of variants of structures (such as semi-group, monoid, or abelian groups) by specifying the signature separately from the adjectives that correspond to the properties of the structure.

A. Structure Element Interpretation

Every Mizar structure signature called *structured type* is defined as a set of assignments. Each assignment is of the form *sel* $\to$ *spec*, where *sel* is a unique structure element label (called selector in the Mizar language) and *spec* is the specification of the type of the respective element of the structure. The signature of a group is the `addLoopStr` structure. It is specified in MML as follows:

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

where for example `addF -> BinOp of the carrier` denotes that + is a binary operation on the field `carrier`. The list of structures given in parentheses immediately after the struct keyword, namely `ZeroStr`, `addMagma` are the names of previously defined structures which contain the element `0` (`ZeroStr`) and a set with the binary operator (`addMagma`) respectively.

An Isabelle formalization of a structure type gives rise to a structure prototype. Each instance of the prototype will be a partial function, with the value corresponding to the selector having the respective type specified in the structure prototype. Definitions of this kind, even if common in informal practice, contain a recursive call. The specification can refer to other parts of the structure (in the above example `addF` in `addLoopStr` is a binary operation of the `carrier`). To specify this in Isabelle we further need a meta-level function which for a given object of structured type and a selector as arguments returns the term present in the object:

```
definition TheSelectorOf (the _ of _ 190) where
  func the sel of Term → object means λit.
    for T be object st [sel, T] in Term holds it = T
```

In order to use such functions in the context of structures, the actual specifications cannot be simply types, but rather functions that for a given object of a structured type as an argument returns the type. In particular, the `addF` element specification needs to be defined as `λS. BinOp-of the carrier of S`. To achieve a more Mizar like formulation `addF -> BinOP-of′ the′ carrier` we further introduce abbreviations for the types with arguments:

```
abbreviation TheS (the'' _) where
  TheS ≡ λselector Term. the selector of Term
abbreviation BinOp_of (BinOp-of'' _) where
  BinOp-of′ X ≡ λit. BinOp-of X(it)
```

This allows representing all assignments of the form $selector \to specification$ as a unary predicate (corresponding to the Isabelle definitions of attributes) which describes all partial functions that are the instances of the structure prototype. To allow the computation of the selector of it we add the condition that the selector is in its domain.

```
definition field (_ → _ 91) where
  sel → spec ≡ define_attr (λit.
    the sel of it be spec(it) & sel in dom it)
```

We can finally define actual structure prototypes. A new structure prototype in Isabelle corresponds to a Mizar mode (non-empty type) which is a partial function that satisfies all the constraints specified in the fields:

```
abbreviation(input) struct (struct _ _ [10,10] 10)
  where struct Name Fields ≡
    (Name ≡ define_mode(λit.
      it be Function & it is Fields))
```

The original `addLoopStr` can now be fully formally specified, using a syntax that is very similar to the Mizar original, while at the same time allowing a complete certification:

```
definition struct addLoopStr
  (# carrier → set′;
    addF → BinOp-of′ the′ carrier;
    ZeroF → Element-of′ the′ carrier #)
```

B. Non-emptiness of Structure Types

A definition of a structure prototype in Mizar provides not only the information about the types of the elements described by the signature but also ensures that there is at least one element of the structure type. For this, the Mizar checker verifies that all the defined structure specifications are non-empty. In Isabelle, we need to actually give a formal proof that the structure exists. We can achieve this by using the Hilbert choice operator ϵ, providing for each assignment of the form $selector \to specification$ the pair $\langle selector, \epsilon(specification)\rangle$. In case of the considered `addLoopStr` structure prototype, we can use:

```
term {[carrier, the set]}∪
      {[addF, the BinOp-of the set]}∪
      {[ZeroF, the Element-of the set]}
```

Proofs of non-emptiness require a lot of effort especially with structures with more elements (some structures have as much as 12 elements). Such proofs ignore the non-emptiness proofs from the structure ancestors. We will, therefore, propose in the next Subsection IV-C a mechanism able to extend an object by the missing elements possibly changing their order. This is desired because a structure definition also implicitly defines:

- the attribute strict which means that the domain of the object contains precisely the selectors indicated in the definition and no other selectors;
- the restriction operation which restricts an object to its strict domain.

Therefore, we provide a scheme for defining the correctness of structures. We present this lemma as well as the majority of lemmas in the paper without proofs which can be found in the development.

```
lemma struct_scheme:
assumes df:
  S ≡ define_mode(λit. it be Function & it is Fields)
 and ex:
   ex X be Function st X is Fields & dom X = D
 and monotone: for X1 be Function st X1 is Fields
             holds D ⊆ dom X1
 and restriction: for X1 be Function st X1 is Fields
             holds X1|D is Fields
shows (x be S iff (x be Function & x is Fields)) &
       Ex (λx. x be S) & domain_of S = D &
       (for X be S holds
          the_restriction_of X to S be (strict S) || S)
```

which given the subproofs for the existence condition `ex` and monotonicity `monotone` allows showing the correctness of the `domain_of` definition (i.e. existance and uniqueness) for the defined structure S, additionally deriving the equality `domain_of S = D`, where

```
definition domain_of (domain'_of _ 200) where
  func domain_of M → set means
    (λit. (ex X be M st it = dom X) &
          (for X be M holds it ⊆ dom X))
```

Furthermore, by proving the restriction definition to the equality, we get the information that `X | domain_of S` is of the structured type of `S` which has the attribute `strict`, if `X` is of the structured type of `S`, which completes the definition

```
definition restriction (the'_restriction'_of _ to _ 190)
where
  func the_restriction_of X to Struct →
          strict Struct || Struct equals
    X | domain_of Struct
```

where the definition of strict is as follows

```
definition strict :: Mode ⇒ Attr (strict _ 200) where
  attr strict M means
    (λX. X be M & dom X = domain_of M)
```

C. Recursive Structure Correctness Conditions

As discussed in the previous section, the `struct_scheme` lemma assumptions can be used to show the non-emptiness of a defined structure type `S`. However, the assumptions about each ancestor `A` of the structure are insufficient to be usable as part of the proof for `S`. In particular, there is no condition that would correspond to `restriction`, which could give the information which extensions of `A` (the extensions of the function that describe the object instance) satisfy all the assignments of `A`. For this reason, we propose a version of the assumption in `struct_scheme` with the additional fourth correctness condition

```
definition struct_well_defined :: Attr ⇒ Set ⇒ o
         ( _ well defined on _[10,10] 200)
where
  Fields well defined on D ≡
     (ex X be Function st X is Fields & dom X=D)
   & (for X1 be Fields||Function holds D ⊆ dom X1)
   & (for X1 be Fields||Function holds X1|D is Fields)
   & (for X1 be Fields||Function, X2 be Function st
        D ⊆ dom X1 & X1 ⊆ X2 holds X2 is Fields)
```

This allows a weaker defining lemma assumption

```
lemma struct_well_defined:
assumes df:
  S ≡ define_mode(λit. it be Function & it is Fields)
 and well: Fields well defined on D
shows (x be S iff (x be Function & x is Fields)) &
    Ex (λx. x be S) & domain_of S = D &
    (for X be S holds
       (the_restriction_of X to S) be (strict S) || S)
```

With these modifications, we can show that an existing list of assignments specified for the domain `D` can be modified by adding a new `selector → specification` pair assuming that the `selector` is not present in `D` so far, and the `specification` uses the selectors of `D`. An example lemma that allows extending a structure is:

```
theorem Fields_add_argM1:
assumes Fields well defined on D
 and selector_1 in D
 and not (selector in D)
 and for X1 be Fields||Function holds
          ex S be M1 (the selector_1 of X1) st True
shows
 Fields | (selector → λS. M1 (the selector_1 of S))
        well defined on D ∪ {selector}
```

This can now be practically used to simplify the non-emptiness proof of `addLoopStr` using the previous proof of the well-definedness of `addMagma` over the set {`carrier`}∪ {`addF`} as follows:

```
lemma addLoopStr_well:
  (# carrier → set';
```

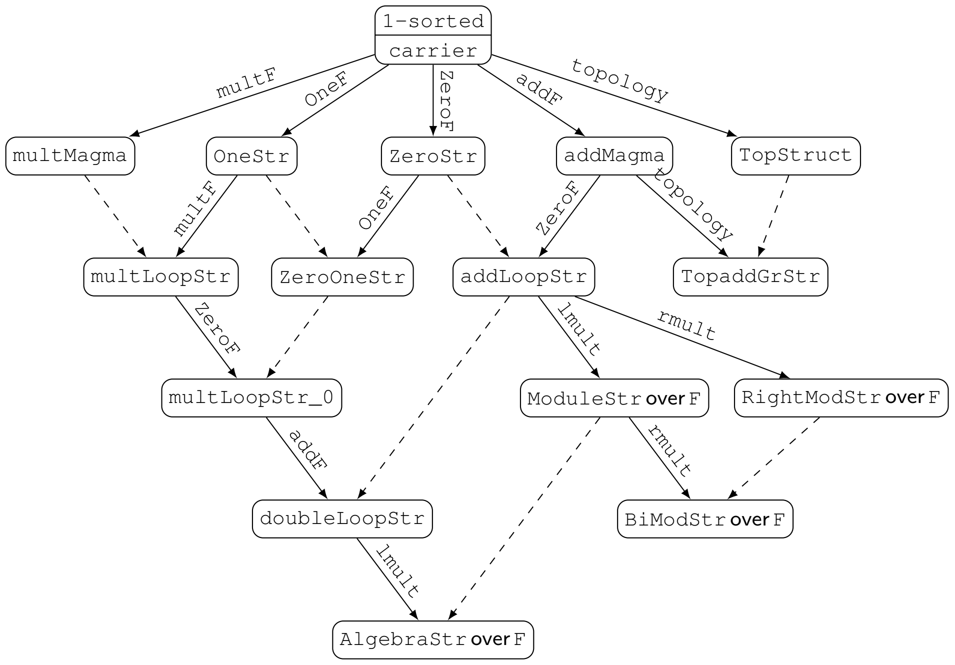

Figure 1. Net of the basic algebraic structures in the Mizar Mathematical Library following [37]. The presented ones have already been covered in our formalization. The arrow captions indicate the added selectors. Solid lines indicate the use of the ancestor structure in the well-definedness proofs, and dashed lines indicate that the ancestor structure is omitted in the proofs.

```
        addF → BinOp-of′ the′ carrier;
        ZeroF → Element-of′ the′ carrier #)
      well defined on {carrier} ∪ {addF} ∪ {ZeroF}
  proof (rule Fields_add_argM1[OF addMagma_well])
    show carrier in {carrier} ∪ {addF}
      by (simp add:string)
    show not ZeroF in {carrier} ∪ {addF}
      by (simp add:string)
    show for X1 be addMagma_fields||Function holds
          ex it be Element-of-struct X1 st True
    proof
      fix X1 assume X1 be addMagma_fields||Function
      hence the carrier of X1 be set using field by auto
      thus ex it be Element-of-struct X1 st True
        using subset_1_def_1 by blast
    qed
  qed
```

where the proof only needs to use the non-emptiness of the type `Element of set`. Furthermore, the fact that the `carrier` is a member of {`carrier`} ∪ {`addF`}, as well as the fact that `ZeroF` is not a member of {`carrier`}∪{`addF`} can both be handled completely automatically by the simplifier in all such proofs.

The well-definedness of addLoopStr does not need to rely on that of the `addMagma` ancestor. One could instead extend the list of assignments of `ZeroStr` by `addF → BinOp-of′ the′ carrier` and change the order. For this purpose we provide the lemma:

```
  theorem well_defined_order:
    assumes ⋀X. X is Fields1 iff X is Fields2
      and Fields1 well defined on D1
    shows Fields2 well defined on D1
```

The components described above are sufficient to define all the MML structures (the basic ones are presented in Fig. 1). The construction follows the recursive element addition approach.

Even if the `addLoopStr` proof refers to its ancestors, the inheritance information is not provided again by `structSchemeWell`. The Mizar system allows indicating this information directly in the structure definition by giving a list of all ancestors. In our approach, it is possible to prove a structure inheritance. Such proofs can be always automatically performed by the simplifier.

```
  theorem addLoopStr_inheritance:
    assumes X be addLoopStr
    shows X be addMagma & X be ZeroStr
      using addLoopStr addMagma ZeroStr assms
      by simp
```

V. Net of Basic Algebraic Structures

Mizar structures together with the inheritance mechanisms significantly facilitate the formalization of computer systems and various domains of mathematics, as well as combining them. For this reason, structures are a challenge for our project, especially the struct_0 article which defines the elementary structures and their operations.

MML contains today 168 structure signatures. Structure signatures form a net because of multiple inheritances. Nevertheless, 135 of the signatures inherit from 1-sorted, namely

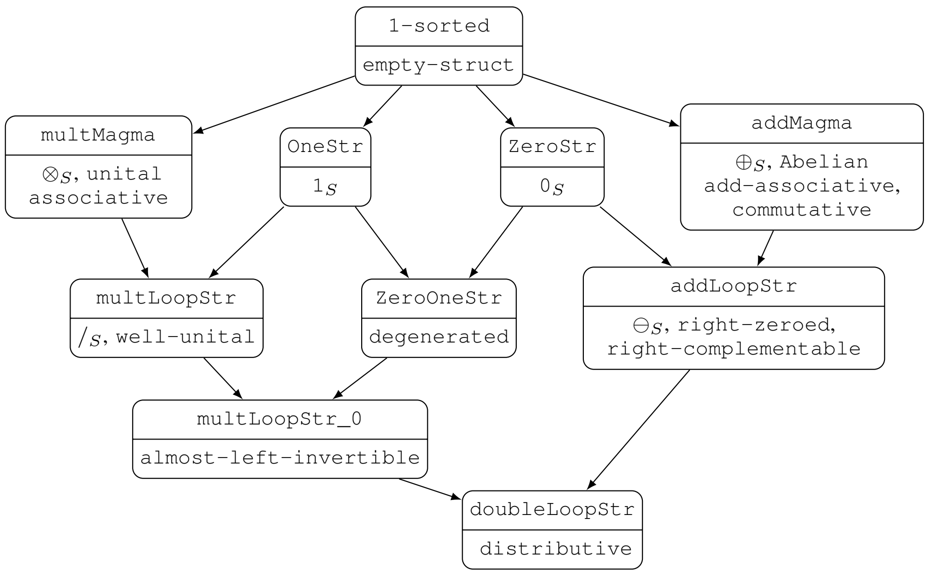

Figure 2. The net of `doubleLoopStr` structure signature ancestors in the MML. For each node, the adjectives required to define a field as well as the unary and binary operations performed on the elements, are listed below.

the signature of structures that contain a carrier which includes some examples in most developed domains of mathematics in the MML, such as algebra, topology, and the theory of lattices. The basic structures are depicted in Fig. 1. These 15 signatures are the direct ancestors of 57 other structure signatures in the MML. Furthermore, these 15 are directly used to define 293 Mizar types (this includes non-expandable types [36]), 291 attributes, 962 functors, and 91 predicates.

As structure signatures are mostly used with adjectives, we reformalize the chosen structures along with their attributes to demonstrate that they can be used efficiently in subsequent proofs. In the paper we focus on `doubleLoopStr`, defined in the MML article `ALGSTR_0`:

```
struct (addLoopStr, multLoopStr_0)
   doubleLoopStr (#
      carrier -> set,
      addF -> BinOp of the carrier,
      ZeroF -> Element of the carrier,
      multF -> BinOp of the carrier,
      OneF -> Element of the carrier #)
```

that inherits from both `addLoopStr` and `multLoopStr_0`, i.e., the signatures of additive and multiplicative groups, respectively. The `doubleLoopStr` structure is also the direct ancestor of the signature or `ModuleStr` over `F` used in vector space domains, where F represents the set of scalar values. A correct definition of `ModuleStr` over `F` permits us to verify our model and our approach for structures parametrized by other structures.

A. Field Formalization

For our formalization of the signature and basic properties of fields, it was necessary to adapt 20 MML articles. Our reformalization focused on the articles `STRUCT_0`, `GROUP_1`, `RLVECT_1`, `ALGSTR_0`, `VECTSP_1`, which define all the ancestors of a field (`doubleLoopStr`), and the main adjectives used in the field definitions, as well as the basic binary and unary operations. In particular we completely cover `ALGSTR_0` in Isabelle/Mizar, which includes 43 functor and predicate definitions (including 13 correctness condition proofs), 72 registrations: non-emptiness of types and relations between groups of adjectives defined on structures, and 6 signatures including

```
definition
 struct doubleLoopStr (#
   carrier → set';
   addF → BinOp-of' the' carrier;
   multF → BinOp-of' the' carrier;
   OneF → Element-of' the' carrier;
   ZeroF → Element-of' the' carrier #)
```

The signature can be used for more complex algebraic structures by extending it by appropriate adjectives. In particular we exactly imitate the MML definitions:

```
abbreviation
 Ring ≡ Abelian | add-associative | right-zeroed |
         right-complementable | associative |
         well-unital | distributive |
         non empty-struct || doubleLoopStr
abbreviation
 SkewField ≡ non degenerated |
                 almost-left-invertible || Ring
abbreviation
  Field ≡ commutative || SkewField
```

The adjectives used in the above definitions have been specified for the various ancestors of `doubleLoopStr` (see Fig. 2). Such definitions have been moved to earliest possible structures as part of the MML refactoring. This allows easy

```
definition
    let S be ZeroStr;
    func 0.S -> Element of S equals
        the ZeroF of S;
end;

definition
    let S be OneStr;
    func 1.S -> Element of S equals
        the OneF of S;
end;

definition
    let M be addMagma;
    let x,y be Element of M;
    func x + y -> Element of M equals
        (the addF of M).(x,y);
end;

definition
    let M be multMagma;
    let x,y be Element of M;
    func x * y -> Element of M equals
        (the multF of M).(x,y);
end;
```

```
definition struct_0_def_6_prefix ( 0_ [1000] 99) where
  func 0_S → Element-of-struct S  equals
     the ZeroF of S
schematic_goal struct_0_def_6:
  assumes S be ZeroStr shows ?X

definition struct_0_def_7_prefix (1_ [1000] 99) where
  func 1_S → Element-of-struct S  equals
     the OneF of S
schematic_goal struct_0_def_7:
  assumes S be OneStr shows ?X

definition algstr_0_def_1 (_ ⊕_ _ [66,1000,67] 66) where
  func x ⊕_M y → Element-of-struct M equals
     (the addF of M) . (| x , y |)
schematic_goal algstr_0_def_1:
  assumes M be addMagma & x be Element-of-struct M
         & y be Element-of-struct M shows ?X

definition algstr_0_def_18 (_ ⊗_ _ [96, 1000, 97] 96) where
  func x ⊗_M y → Element-of-struct M equals
     (the multF of M) . (| x , y |)
schematic_goal algstr_0_def_18:
  assumes M be multMagma & x be Element-of-struct M
         & y be Element-of-struct M shows ?X
```

Figure 3. Selected definitions of highlighted elements and binary operations in doubleLoopStr originally formulated in the MML and their Isabelle/Mizar reformulations.

```
definition
    let M be addLoopStr, x be Element of M;
    assume A1: x is left_complementable
                    right_add-cancelable;
    func -x -> Element of M means
        it + x = 0.M;
end;

definition
    let M be multLoopStr, x be Element of M;
    assume A1: x is left_invertible
                    right_mult-cancelable;
    func /x -> Element of M means
        it * x = 1.M;
end;
```

```
definition algstr_0_def_13 (⊖_ _ [1000, 86] 87) where
  assume x is left-complementable_M | right-add-cancelable_M
   func ⊖_M x → Element-of-struct M means
     (λit. it ⊕_M x = 0_M)
schematic_goal algstr_0_def_13:
  assumes M be addLoopStr
          x be Element-of-struct M shows ?X

definition algstr_0_def_30 ('/_ _ [1000, 99] 98) where
  assume x is left-invertible_M | right-mult-cancelable_M
   func /_M x → Element-of-struct M means
     (λit. it ⊗_M x = 1_M)
schematic_goal algstr_0_def_30[rule_format]:
  assumes M be multLoopStr
          x be Element-of-struct M shows ?X
```

Figure 4. Selected conditional definitions of unary operations from ALGSTR_0 originally formulated in the MML and their Isabelle/Mizar reformulation.

import of developed theories, which we want to now evaluate in Isabelle/Mizar. Consider the theory of additive groups. It is mostly defined over the structure addLoopStr with the adjectives add-associative, right_zeroed right_-complementable. As addLoopStr is an ancestor of the doubleLoopStr signature, this set of adjectives is a subset of that used for example for rings, therefore, properties of additive group can be used in the context of rings in the MML.

Moreover, Mizar allows the use of functors defined on ancestors with arguments of further types. We show the definitions of the selected elements and operations of doubleLoopStr, namely 0, 1, +, and * are defined in Fig. 3. In Mizar, the patterns of the symbols are given in previously specified dictionaries, while in Isabelle these need to be given in the definition block. Furthermore, the definition is split into two parts: the pattern without the argument types and the definition theorems. This allows reducing the number of visible arguments corresponding to hidden arguments in Mizar, as well as allows interpreting conditional definitions (see Fig. 4). The conditions need to appear in the pattern in the Isabelle/Mizar approach. More details are given in the Mizar_defs theory file.

As case studies, we show that the proposed way to model structures and their inheritance is sufficient not only to define attributes and functors but also is adequate for imitating Mizar-style formalization. For this purpose, we reformalize (so far manually) selected theorems that concern, e.g. the additive and multiplicative groups, and use them in the context of doubleLoopStr. Here we show a single selected proof of the statement that the product of two elements is zero

```
theorem Th12:
   for F being add-associative right_zeroed
      right_complementable associative commutative
      well-unital almost_left_invertible
      distributive non empty doubleLoopStr,
      x,y being Element of F holds
   x * y = 0.F iff x = 0.F or y = 0.F
proof

   let F be add-associative right_zeroed
      right_complementable associative commutative
      well-unital almost_left_invertible distributive
      non empty doubleLoopStr,
      x be Element of F,
      y be Element of F;










   x * y = 0.F implies x = 0.F or y = 0.F
   proof
      assume A1: x * y = 0.F;
      assume A2: x <> 0.F;
      x" * (0.F) = x" * x * y by A1,GROUP_1:def 3

                 .= (1.F) * y by A2,Def10
                 .= y;
      hence thesis;

   end;
   hence thesis;


end;
```

```
theorem vectsp_1_th_12:
 for F being add-associative | right-zeroed |
       right-complementable | associative | commutative |
       well-unital | almost-left-invertible |
       distributive | non empty-struct || doubleLoopStr,
     x,y being Element-of-struct F holds
  x ⊗_F y = 0_F iff x = 0_F or y = 0_F
proof(intro ballI)
 fix F x y
 assume T:F be add-associative | right-zeroed |
        right-complementable | associative | commutative |
        well-unital | almost-left-invertible |
        distributive | non empty-struct || doubleLoopStr
       x be Element-of-struct F
       y be Element-of-struct F
 hence A:F be multLoopStr_0 F be multLoopStr F be ZeroStr
 using doubleLoopStr multLoopStr_0 multLoopStr ZeroStr by auto
 have I: x''^F be Element-of-struct F
   using algstr_0_def_30[of F x] T A by auto
 have Z: 0_F be zero _F || Element-of-struct F
   using struct_0_def_12_a[of F] struct_0_def_6[of F] A by auto
 have x ⊗_F y = 0_F implies x = 0_F or y = 0_F
   proof(rule impI,rule disjCI2)
     assume A1:x ⊗_F y = 0_F
     assume A2:x <> 0_F
     have x''^F ⊗_F 0_F = x''^F ⊗_F x ⊗_F y
       using A1 group_1_def_3a T I by auto
     also have ... = 1_F ⊗_F y using A2 vectsp_1_def_10 T by auto
     also have ... = y using vectsp_1_reduce_2 T A by auto
     finally show y = 0_F using vectsp_1_reduce_3[OF _ I Z]
       T vectsp_1_cl_20 by auto
 qed
 thus x ⊗_F y = 0_F iff x = 0_F or y = 0_F using
   vectsp_1_reduce_4[OF _ T(3) Z] vectsp_1_reduce_3[OF _ T(2) Z]
   T vectsp_1_cl_20 by auto
qed
```

Figure 5. A property of fields originally formulated in the VECTSP_1 article and its Isabelle/Mizar reformulation.

if and only if at least one of them is zero (see Fig. 5). Note that the justification of steps refer to theories developed for multLoopStr_0, multLoopStr, ZeroStr structures (see steps that use label A in justifications). Additionally, each such justification uses some of the attributes indicated in the step labeled by T.

We make use of Isabelle features to model the proofs in such a way that the Isabelle/Mizar language can be as close as possible to the Mizar one. This simplifies the comparison of both proofs. The proofs in our certification contain more steps than the Mizar ones. This is mainly due to the lack of the Mizar automation in our system, e.g., type inference, equational calculus [38], definitional expansions [39]. Additionally, we still have to directly indicate the background information, such as registrations, that are processed automatically by Mizar. We are currently working on mechanisms that would reduce the numbers of additional steps required.

VI. Conclusion

We have presented the progress in our project aiming to independently certify Mizar proofs in the Isabelle logical framework. The proposed recursive approach to structures allows more readable proofs of well-definedness, as well as a more concise way to specify structure inheritance. We verified the provided mechanisms by reformalizing the complete Mizar article defining basic algebraic structures ALGSTR_0, as well as parts of several articles that define and prove properties of structures contain other structures as fields. The experiments confirm that the proposed approach is convenient for proving structure properties.

The Isabelle/Mizar formalization currently includes 31 theorems, 97 registrations including 6 reductions, 86 definitions where 37 of them required Mizar-style justifications and 4 redefinitions concerning MML structures. The total size of the development is 416 kB and 9742 lines of code. It is available at:

http://cl-informatik.uibk.ac.at/cek/fedcsis2017/

A. Future Work

Our certification work has so far focused on the foundations, definitions, and registrations available in the Mizar language. A natural next step would be to allow an implicit use of the background knowledge. Without it, redundant steps in reasoning to represent information computed the Mizar type-inference mechanisms have been necessary so far. Furthermore, we plan to translate the whole MML into the

Isabelle/Mizar environment in an automated way and with the help of automatically finding related concepts between logics [40], as well as by improving the currently available Isabelle automation for Mizar [41] we hope to cross-verify large parts of the translated MML in Isabelle which is also one of the important steps in the creation of a combined formal library spanning multiple foundations and systems [42].

Acknowledgements

This work has been supported by the OeAD Scientific & Technological Cooperation with Poland grant PL 03/2016.

REFERENCES

[1] X. Leroy, "Formal verification of a realistic compiler," *Commun. ACM*, vol. 52, no. 7, pp. 107–115, 2009. doi: 10.1145/1538788.1538814

[2] G. Klein, J. Andronick, K. Elphinstone, T. C. Murray, T. Sewell, R. Kolanski, and G. Heiser, "Comprehensive formal verification of an OS microkernel," *ACM Trans. Comput. Syst.*, vol. 32, no. 1, p. 2, 2014. doi: 10.1145/2560537

[3] J. Harrison, "Floating-Point Verification," *J. UCS*, vol. 13, no. 5, pp. 629–638, 2007. doi: 10.3217/jucs-013-05-0629

[4] T. C. Hales, M. Adams, G. Bauer, D. T. Dang, J. Harrison, T. L. Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T. T. Nguyen, T. Q. Nguyen, T. Nipkow, S. Obua, J. Pleso, J. Rute, A. Solovyev, A. H. T. Ta, T. N. Tran, D. T. Trieu, J. Urban, K. K. Vu, and R. Zumkeller, "A formal proof of the Kepler conjecture," *Forum of Mathematics, Pi*, vol. 5, 2017. doi: 10.1017/fmp.2017.1

[5] G. Bancerek, C. Bylinski, A. Grabowski, A. Korniłowicz, R. Matuszewski, A. Naumowicz, K. Pąk, and J. Urban, "Mizar: State-of-the-art and Beyond," in *Intelligent Computer Mathematics - International Conference, CICM 2015*, ser. LNCS, M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge, Eds., vol. 9150. Springer, 2015. doi: 10.1007/978-3-319-20615-8_17 pp. 261–279.

[6] J. Alama, M. Kohlhase, L. Mamane, A. Naumowicz, P. Rudnicki, and J. Urban, "Licensing the Mizar Mathematical Library," in *Proc. 10th International Conference on Intelligent Computer Mathematics (CICM 2011)*, ser. LNCS, J. H. Davenport, W. M. Farmer, J. Urban, and F. Rabe, Eds., vol. 6824. Springer, 2011. doi: 10.1007/978-3-642-22673-1_11 pp. 149–163.

[7] M. Wenzel, L. C. Paulson, and T. Nipkow, "The Isabelle framework," in *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008*, ser. LNCS, O. A. Mohamed, C. A. Muñoz, and S. Tahar, Eds., vol. 5170. Springer, 2008. doi: 10.1007/978-3-540-71067-7_7 pp. 33–38.

[8] J. C. Blanchette, M. Haslbeck, D. Matichuk, and T. Nipkow, "Mining the Archive of Formal Proofs," in *Intelligent Computer Mathematics (CICM 2015)*, ser. LNCS, M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge, Eds., vol. 9150. Springer, 2015. doi: 10.1007/978-3-319-20615-8_1 pp. 3–17.

[9] Y. Bertot, "A Short Presentation of Coq," in *Theorem Proving in Higher Order Logics (TPHOLs 2008)*, ser. LNCS, O. A. Mohamed, C. A. Muñoz, and S. Tahar, Eds., vol. 5170. Springer, 2008. doi: 10.1007/978-3-540-71067-7_3 pp. 12–16.

[10] L. Cruz-Filipe, H. Geuvers, and F. Wiedijk, "C-CoRN, the Constructive Coq Repository at Nijmegen," in *Mathematical Knowledge Management (MKM'04)*, ser. LNCS, A. Asperti, G. Bancerek, and A. Trybulec, Eds., vol. 3119. Springer, 2004. doi: 10.1007/978-3-540-27818-4_7 pp. 88–103.

[11] G. Gonthier and A. Mahboubi, "An introduction to small scale reflection in Coq," *J. Formalized Reasoning*, vol. 3, no. 2, pp. 95–152, 2010. doi: 10.6092/issn.1972-5787/1979

[12] A. Grabowski and T. Mitsuishi, "Formalizing Lattice-Theoretical Aspects of Rough and Fuzzy Sets," in *Rough Sets and Knowledge Technology – 10th International Conference, RSKT 2015, held as part of the International Joint Conference on Rough Sets, IJCRS 2015, Tianjin, China, November 20–23, 2015, Proceedings*, 2015. doi: 10.1007/978-3-319-25754-9_31 pp. 347–356.

[13] L. C. Paulson, "Isabelle: The next 700 theorem provers," in *Logic and Computer Science (1990)*, P. Odifreddi, Ed., 1990, pp. 361–386.

[14] C. Kaliszyk, K. Pąk, and J. Urban, "Towards a Mizar Environment for Isabelle: Foundations and Language," in *Proc. 5th Conference on Certified Programs and Proofs (CPP 2016)*, J. Avigad and A. Chlipala, Eds. ACM, 2016. doi: 10.1145/2854065.2854070 pp. 58–65.

[15] C. Kaliszyk and K. Pąk, "Presentation and Manipulation of Mizar Properties in an Isabelle Object Logic," in *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings*, ser. LNCS, H. Geuvers, M. England, O. Hasan, F. Rabe, and O. Teschke, Eds., vol. 10383. Springer, 2017. doi: 10.1007/978-3-319-62075-6_14 pp. 193–207.

[16] C. Schürmann, "The Twelf Proof Assistant," in *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009*, ser. LNCS, S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, Eds., vol. 5674. Springer, 2009. doi: 10.1007/978-3-642-03359-9_7 pp. 79–83.

[17] F. Rabe, "A logical framework combining model and proof theory," *Mathematical Structures in Computer Science*, vol. 23, no. 5, pp. 945–1001, 2013. doi: 10.1017/S0960129512000424

[18] M. Wenzel, "Isar - A Generic Interpretative Approach to Readable Formal Proof Documents," in *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99*, ser. LNCS, Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, Eds., vol. 1690. Springer, 1999. doi: 10.1007/3-540-48256-3_12 pp. 167–184.

[19] M. Wenzel and F. Wiedijk, "A Comparison of Mizar and Isar," *J. Autom. Reasoning*, vol. 29, no. 3-4, pp. 389–411, 2002. doi: 10.1023/A:1021935419355

[20] K. Pąk, "Readable Formalization of Euler's Partition Theorem in Mizar," in *Intelligent Computer Mathematics – International Conference, CICM 2015, Washington, DC, USA, July 13–17, 2015, Proceedings*, 2015. doi: 10.1007/978-3-319-20615-8_14 pp. 211–226.

[21] K. Pąk, "Automated Improving of Proof Legibility in the Mizar System," in *Intelligent Computer Mathematics – International Conference, CICM 2014, Coimbra, Portugal, July 7–11, 2014. Proceedings*, 2014. doi: 10.1007/978-3-319-08434-3_27 pp. 373–387.

[22] K. Pąk, "Improving Legibility of Formal Proofs Based on the Close Reference Principle is NP-hard," *Journal of Automated Reasoning*, vol. 55, no. 3, pp. 295–306, 2015. doi: 10.1007/s10817-015-9337-1

[23] J. Harrison, "A Mizar Mode for HOL," in *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, ser. LNCS, J. von Wright, J. Grundy, and J. Harrison, Eds., vol. 1125. Springer, 1996. doi: 10.1007/BFb0105406 pp. 203–220.

[24] C. Kaliszyk and F. Wiedijk, "Merging Procedural and Declarative Proof," in *Types for Proofs and Programs, International Conference, TYPES 2008*, ser. LNCS, S. Berardi, F. Damiani, and U. de'Liguoro, Eds., vol. 5497. Springer, 2008. doi: 10.1007/978-3-642-02444-3_13 pp. 203–219.

[25] D. Syme, "Three Tactic Theorem Proving," in *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Nice, France, September, 1999, Proceedings*, ser. LNCS, Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin-Mohring, and L. Théry, Eds., vol. 1690. Springer, 1999. doi: 10.1007/3-540-48256-3_14 pp. 203–220.

[26] F. Wiedijk, "Mizar Light for HOL Light," in *Theorem Proving in Higher Order Logics, 14th International Conference, TPHOLs 2001*, ser. LNCS, R. J. Boulton and P. B. Jackson, Eds., vol. 2152. Springer, 2001. doi: 10.1007/3-540-44755-5_26. ISBN 3-540-42525-X pp. 378–394.

[27] ——, "A Synthesis of the Procedural and Declarative Styles of Interactive Theorem Proving," *Logical Methods in Computer Science*, vol. 8, no. 1, 2012. doi: 10.2168/LMCS-8(1:30)2012

[28] P. Corbineau, "A Declarative Language for the Coq Proof Assistant," in *Types for Proofs and Programs, International Conference, TYPES 2007*, ser. LNCS, M. Miculan, I. Scagnetto, and F. Honsell, Eds., vol. 4941. Springer, 2007. doi: 10.1007/978-3-540-68103-8_5. ISBN 978-3-540-68084-0 pp. 69–84.

[29] S. Jaśkowski, "On the rules of suppositions," *Studia Logica*, vol. 1, 1934.

[30] J. Urban, "MPTP 0.2: Design, implementation, and initial experiments," *J. Autom. Reasoning*, vol. 37, no. 1–2, pp. 21–43, 2006. doi: 10.1007/s10817-006-9032-3

[31] C. Kaliszyk and J. Urban, "MizAR 40 for Mizar 40," *J. Autom. Reasoning*, vol. 55, no. 3, pp. 245–256, 2015. doi: 10.1007/s10817-015-9330-8

[32] O. Kunčar, "Reconstruction of the Mizar type system in the HOL Light system," in *WDS Proceedings of Contributed Papers: Part I – Mathematics and Computer Sciences*, J. Pavlu and J. Safrankova, Eds. Matfyzpress, 2010, pp. 7–12.

[33] A. Grabowski, A. Korniłowicz, and A. Naumowicz, "Mizar in a Nutshell," *J. Formalized Reasoning*, vol. 3, no. 2, pp. 153–245, 2010. doi: 10.6092/issn.1972-5787/1980

[34] M. Iancu, M. Kohlhase, F. Rabe, and J. Urban, "The Mizar Mathematical Library in OMDoc: Translation and applications," *J. Autom. Reasoning*, vol. 50, no. 2, pp. 191–202, 2013. doi: 10.1007/s10817-012-9271-4

[35] L. C. Paulson, "Set theory for verification: I. From foundations to functions," *J. Autom. Reasoning*, vol. 11, no. 3, pp. 353–389, 1993. doi: 10.1007/BF00881873

[36] A. Grabowski, A. Korniłowicz, and A. Naumowicz, "Four Decades of Mizar," *Journal of Automated Reasoning*, vol. 55, no. 3, pp. 191–198, October 2015. doi: 10.1007/s10817-015-9345-1

[37] A. Grabowski, A. Korniłowicz, and C. Schwarzweller, "On Algebraic Hierarchies in Mathematical Repository of Mizar," in *Proc. Federated Conference on Computer Science and Information Systems (FedCSIS 2016)*, M. Ganzha, L. A. Maciaszek, and M. Paprzycki, Eds., 2016. doi: 10.15439/2016F520 pp. 363–371.

[38] A. Grabowski, A. Korniłowicz, and C. Schwarzweller, "Equality in Computer Proof-Assistants," in *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems*, ser. Annals of Computer Science and Information Systems, M. Ganzha, L. A. Maciaszek, and M. Paprzycki, Eds., vol. 5. IEEE, 2015. doi: 10.15439/2015F229 pp. 45–54.

[39] A. Korniłowicz, "Definitional expansions in Mizar," *Journal of Automated Reasoning*, vol. 55, no. 3, pp. 257–268, October 2015. doi: 10.1007/s10817-015-9331-7. [Online]. Available: http://dx.doi.org/10.1007/s10817-015-9331-7

[40] T. Gauthier and C. Kaliszyk, "Matching concepts across HOL libraries," in *Proc. of the 7th Conference on Intelligent Computer Mathematics (CICM'14)*, ser. LNCS, S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban, Eds., vol. 8543. Springer, 2014. doi: 10.1007/978-3-319-08434-3_20 pp. 267–281.

[41] C. Kaliszyk and J. Urban, "Learning-assisted Theorem Proving with Millions of Lemmas," *Journal of Symbolic Computation*, vol. 69, pp. 109–128, 2015. doi: 10.1016/j.jsc.2014.09.032

[42] P. Corbineau and C. Kaliszyk, "Cooperative Repositories for Formal Proofs," in *Towards Mechanized Mathematical Assistants, 14th Symposium, Calculemus 2007, 6th International Conference, MKM 2007, Hagenberg, Austria, June 27-30, 2007, Proceedings*, ser. LNCS, M. Kauers, M. Kerber, R. Miner, and W. Windsteiger, Eds., vol. 4573. Springer, 2007. doi: 10.1007/978-3-540-73086-6_19 pp. 221–234.