Isabelle Formalization of Set Theoretic Structures and Set Comprehensions ⋆

Cezary Kaliszyk[1] and Karol Pąk[2]

[1] Universität Innsbruck, Austria
cezary.kaliszyk@uibk.ac.at
[2] Uniwersytet w Białymstoku, Poland
pakkarol@uwb.edu.pl

Abstract. Reasoning about computers and programming languages on paper is most often done with set theory, while most proof assistant formalizations of languages and programs use alternative mathematical foundations. One of the few exceptions has been Mizar where the *Simple Concrete Model* of computers has been used to verify programs expressed as abstract programming language instruction sequences. The model uses extended set theory features including structures and Fraenkel set comprehension operators. In this paper we show how to formally specify such objects in the Isabelle object logic implementing the Mizar foundations as definitional extensions. To show the adequacy and usability of the mechanisms, we reformalize a number of Mizar definitions and theorems related to structures and set comprehensions, including both mathematical and programming language examples: groups, machines and properties of computer memory states.

Keywords: Isabelle, Mizar, structure, set comprehension, multiple inheritance

1 Introduction

Proof assistants are today increasingly used to certify software, hardware, as well as mathematical proofs that involve computer programs [10]. One of the earliest proof assistants, Mizar [7], has been developed as a tool to provide a human-oriented environment which would allow proofs to be formally analyzed. The system has already been developed over forty years with its most distinctive features being a proof style that imitates informal mathematical proofs as much as possible [16] and a rich type system that reflects how mathematicians and computer scientists describe dependencies between objects [25]. Such support for formal proofs has given rise to one of the largest libraries of formalized mathematics with many domains not covered in other libraries. One of such domains is the *Simple Concrete Model* (SCM) [17], which introduces a formal model corresponding to random-access Turing machines, their instructions, and

⋆ The paper has been supported by the resources of the Polish National Science Center granted by decision n°DEC-2015/19/D/ST6/01473.

programs and has been considered more realistic for modeling of real computers [21]. The development of SCMs and the proofs of their various properties spans 66 *Mizar Mathematical Library* (MML) articles.

We build upon our recent work which aimed to specify foundations [13], notations [12], automation [14] of the Mizar system in the Isabelle Logical Framework [24]. The most important motivation for the current work is to provide the completely specified set theoretic formalizations of the model of computers, instructions, languages, etc. There are multiple further directions for how we plan to extend that work, as well as multiple reasons for these extensions:

- Specifying Mizar in a logical framework gives the complete semantics of the system specified only on paper so far [6], including the underlying first-order logic variant, the soft type system, definitional mechanisms, and automation mechanisms.
- Despite various efforts [11], the contents of the MML are hard to access for developers of other proof and knowledge management systems. Isabelle's structures can allow experiments with sharing proof techniques and automation across proof assistants.
- Mizar has a large monolithic kernel. Despite the implementers best efforts, bugs in the code can result in incorrect proofs being accepted. This problem can be significantly remedied by certifying proofs across systems.
- In the long run, develop an alternative environment for reverification and development of proofs automatically exported from the MML.

In this work we introduce and develop two components used in Mizar necessary to translate and certify the MML proofs on algebraic structures including the SCM model of computers in Isabelle. The components are Mizar *structures* and Mizar *set comprehension* operators.

Mizar *structures* (also referred to as *aggregates* or *records*) allow grouping multiple other objects together with relations between them into a single entity. This is useful for defining and reasoning about mathematical structures such as rings, fields, and vector spaces. Mizar structures correspond to mechanisms in other proof assistants like the Isabelle type classes [9] or Coq records used to build an algebraic hierarchy [5]. The support for structures is a crucial part of the Mizar language. Structures are built in to the Mizar VERIFIER [6] and they are heavily used in the MML. In fact 74% of the articles in the current MML version 1289 rely directly or indirectly on the article struct_0.

Mizar *set comprehension* operators (referred to as *Fraenkel* in the Mizar literature [6]) allow describing a set of terms whose argument list satisfies a given predicate. Defining set comprehension in a sound and adequate way is an important part of the Mizar foundations, as in any set theory this is where most paradoxes (Russell's paradox and its variants) originate from.

We use the already specified foundations of the Mizar system together with the Tarski-Groethendieck axiomatization, and the first few formalized articles of the MML to formally define Mizar structures and Mizar set comprehension operators. Both can be introduced as definitional extensions, without adding any further axioms.

1.1 Related Work

B-Method [1] has aimed to ease the formalization of programs in a foundation based on set theory, however like Mizar the structures and set comprehension are a part of the system. Similarly, Metamath [20] does not have a built-in notion of structures and focuses on n-tuples instead.

Turing machines have been formalized in Isabelle/HOL [26] allowing reasoning about their behavior in Hoare logic, as well as in Matita [2] focusing on complexity theory. The main approach to formalization of imperative programs in Isabelle is used in Imperative/HOL [4]. This approach was further refined to allow for formalization of programs in separation logic [18].

Algebraic structures were often necessary early in the development of proof assistants. In Isabelle/HOL type classes [9] allow for further control of the polymorphic type system adding mechanisms such as inheritance between types. Various Isabelle automation mechanisms can translate type classes to predicates, which is also how reasoning about algebraic structures with inheritance is usually performed in other HOL-based systems. Proof assistants based on versions of type theory can store objects along with their properties in tuples (or records with named fields). This has been used to build an algebraic hierarchy [5] in Coq or to extend it to topologies as done in Matita [23]. Inheritance for records that can allow for good automation has become an important field with developments including canonical structures and type classes.

Lee and Rudnicki [19] proposed an alternative approach to defining structures without special support in the Mizar system. The main motivation is to make field structures into first-class objects, which allows more convenient reasoning about graphs. The proposed approach directly uses other parts of the Mizar language (including preceding parts of the MML) to define aggregates as Mizar `finite Function`s. This allows defining what it means for an object to have a field, rather than to fix which collections of fields constitute an aggregated object.

The exports of Mizar to ATPs [3] require a specification of the Mizar set comprehensions. The semantics of the exported objects is the same as that in Mizar and in our formalizations, but they are axiomatized rather than defined. We are not aware of any work that specifies the foundations of Mizar in a formal system that would cover structures.

1.2 Contributions and Outline

We give a complete formal specification of Mizar structures formalized in the Isabelle/Mizar object logic (Sect. 3). It supports strict structures (structures that do not include additional fields), domain of a structure (which allows restricting larger structures to smaller ones), and inheritance (which allows extending structures to larger ones) including multiple inheritance.

We formally specify the Mizar Fraenkel set comprehension operator (Sect. 4). Our approach allows defining it as a single meta-level functor, therefore a definitional extension as opposed to a part of the implementation of the CHECKER in Mizar. We further prove a number of properties of this functor.

We reformalize parts of the MML corresponding to the lattice of types focusing on the *simple concrete model* of computers, and show that the defined mechanisms are appropriate and usable to formalize all of Mizar specifics in Isabelle (Sect. 5).

2 Preliminaries

In this section we briefly introduce the Mizar foundations defined as an object logic in the Isabelle framework. For more details see [13].

Four Isabelle types are used to model the Mizar foundations. The type of propositions is already defined by the underlying Isabelle/FOL object logic. The following types are further added: the type of Mizar sets Set and two types used for the Mizar type system: Mode and Attr. Mizar *modes* are the elementary types assigned to all objects. Modes are guaranteed to be non-empty. Mizar *attributes* allow restricting of a given mode or of another attribute. The only attributes considered in this paper will be *adjectives*. Each adjective corresponds to a (parametrizable) predicate on a given type. The type constructed by applying a number of adjectives to a given type corresponds to the elements of the type which satisfy all the adjective-associated predicates. For example the type `non zero natural number` restricts the type (mode) of numbers to both natural ones and those different from zero. For clarity, in the Isabelle formalization the operation that combines attributes will be denoted using a single vertical bar | and the operation of applying attributes to a mode will be denoted using a double one ‖. More information about modes and attributes can be found in [7].

The Isabelle/Mizar object logic introduces constants that allow interacting with the Mizar types, a constant for the choice operator, and five axioms that specify these constants. Two axioms specify what it mean to define a new mode and a new attribute. Two axioms express the meaning of the combinations of attributes with attributes and with modes. The last one axiomatizes the Mizar axiom of choice for non-empty types.

Next, notations that imitate the Mizar text are introduced for the first-order logic symbols: `&`, `or`, `implies`, `for x holds P`, etc. Syntax and helper lemmas are provided to allow defining functions, predicates, and new types in ways similar to that used in Mizar. In particular the definition of a meta-level function `F` which is to return type `T` in Mizar follows the pattern `func F → T equals D` and definitions using the description operator use `means` rather than `equals` and a predicate that the defined object should satisfy. These preliminaries are sufficient to express the Tarski-Grothendieck set theory axiomatization in the same way as in Mizar. Furthermore [13] showed, that it is sufficient to translate all the definitions and theorems from the first few articles of the MML.

3 Structures

Mizar structures are used to define objects that are typically represented as tuples in mathematics. For example the Mizar definition of ring $\langle F, +, 0, \cdot, 1\rangle$

consists of Mizar types assigned to fields in the structure, in particular $+$, $\cdot$ are binary operations on F, and 0, 1 are members of F. To do this, unique identifiers (referred to as a *field selector* or simply *selector* in the Mizar literature) are needed for each tuple element. In case of a ring these identifiers are `carrier`, `addF`, `ZeroF`, `multF`, and `OneF` respectively. The Mizar syntax for the tuple including the above mentioned types is presented on the left. The Isabelle counterpart, which we will define later in this section is presented on the right for comparison (for simplicity inheritance information is omitted here, it will be discussed in Sect. 3.4):

```
struct doubleLoopStr (#
   carrier → set,
   addF → BinOp of the carrier,
   ZeroF → Element of the carrier,
   multF → BinOp of the carrier,
   OneF → Element of the carrier
#)
```

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

The `doubleLoopStr` structure will correspond to a ring only with additional restrictions. Such restrictions are in Mizar introduced using adjectives (see Sect. 2). In particular, a ring in the MML is defined as a `doubleLoopStr` together with nine adjectives, such as `Abelian` and `distributive` with their expected meanings. Certain extensions of a ring, such as a field, will only extend the list of adjectives (for example by `commutative`), which permits all Mizar mechanisms (functors, definitions, theorems) associated with rings to also work with fields. Mizar allows adjectives to be used in the field selector types, which corresponds to structures with resticted values. This is used for SCMs (see Sect. 5.2).

Mizar structures also support inheritance discussed in more detail in Sect. 3.4. Here it is only important to note that inheritance does allow not only ring extensions, but also permits the use of group theory for rings and fields, since the group tuple `multLoopStr` is a sub-tuple of that of `doubleLoopStr`. This means that "being a group" defined for `multLoopStr` must allow tuples that have more than the required selectors. However, there are cases where we want to express the fact that a group has precisely the `multLoopStr` selectors, namely that the tuple does not have any other elements. This is achieved using the Mizar attribute strict that can be applied to any structure, which specifies that only the selectors from that structure are allowed. The need for strict can be illustrated by the following example. Consider the set of all groups over $\mathbb{Z}_3$. This set is finite if and only if we consider strict structures. The net hierarchy of basic algebraic structures in the MML is depicted in Fig. 1.

3.1 Structure Preliminaries

In the Mizar literature the word structure is used both for *structure prototypes* (e.g. the type of rings) and for actual structure instances (e.g. individual objects

that are of the type of rings). We will try to distinguish the two when it is not clear from the context. Structure instances will be represented as set theoretic functions. We will use our Isabelle reformalization of the Mizar set theoretic relations for this purpose. Structure prototype definitions will correspond to schemes of functions, which can be further restricted by the given adjectives.

3.2 Structure Operations

A structure prototype definition will describe functions given as sets of assignments. Each assignment is of the form $x \to y$, where x is a unique label (selector) and y is the specification given to that field of the structure. As the specification may refer to the other parts of the structure (for example the zero in the ring is an element of the carrier), y needs to be a meta-level function which, when given the structure instance as an argument returns the type of that field. We present here the general definitions of the selector and of the single field in a structure in our formalization:

```
definition TheSelectorOf ("the _ of _ " [90,90] 190) where
  "func the selector of Term → object means λit.
    for T be object st ⟨selector,T⟩ in Term holds it = T"

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

With this we can introduce a Mizar-like syntax for structure prototypes (# $\mathtt{f}_1$; ... ; $\mathtt{f}_n$ #), where each field f_i is described by an assignment *sel* $\to$ *spec(it)*. Most basic structure prototypes ignore the argument:

```
definition one_sorted :: "Mode"  ("one-sorted") where
  "struct one-sorted (# carrier → λ_. set #)"
```

We now define the domain of a structure prototype as the minimal set that is contained in the domain of any instance. This allows the following definition to be a global one, however the result makes sense only for a particular prototype.

```
definition domain_of::"Mode ⇒ Set" ("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))"
```

The fact that we know the domain globally also allows creating strict as an attribute. The attribute should consider the domain of the structure type, which may be the last argument after other attributes. Not to restrict the order of attributes, the Isabelle version of strict requires an argument, which repeats the mode. For example strict one-sorted || one-sorted.

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

We can finally introduce the restriction of an instance to a strict structure using the restriction of a function domain denoted with the slash operator.

```
definition the_restriction_of :: "Set ⇒ Mode ⇒ Set"
                    ("the'_restriction'_of _ to _" 90) where
  "func the_restriction_of X to Struct →
    strict Struct || Struct equals X | domain_of Struct"
```

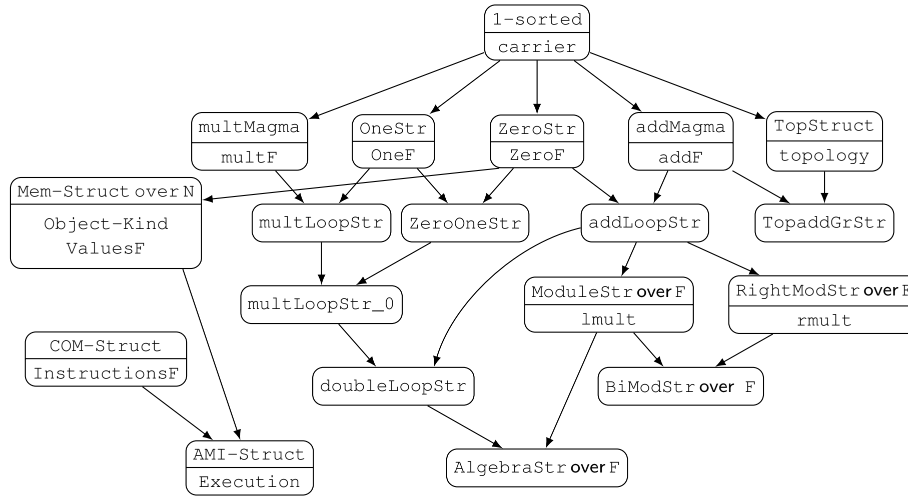

Fig. 1. Net of basic algebraic and computer-related structures in the MML following [8]. The presented ones have already been covered in our formalization. The lower part of each node lists the selectors which are added w.r.t. the inherited ones. *AMI* (Architecture Model for Instructions) is an abstract computer structure parametrized by the data stored in its memory, further detailed in Sect. 5.2.

3.3 Structure Prototype Introduction

To define an actual structure prototype, it is necessary to use an actual set of labels which are pairwise different. In principle strings could be natural for this purpose. However, as we prefer to reduce the required part of the library foundations, we chose to use the set theoretic natural numbers defined by $0 = \{\}$ and $succ(X) = X \cup \{X\}$.

Furthermore, to define an actual structure prototype, it is necessary to show non-emptiness, that is that there exists a structure instance which fulfills the structure prototype conditions. To show such existence, Mizar requires the non-emptiness of all structure field specifications. For this, we use the global choice operator ϵ. For each field (selector $\rightarrow$ specification) we take the pair $\langle$selector, ϵ(specification)$\rangle$. We show that the set of such pairs for all fields of the structure fulfills the prototype conditions (with convenient automation to show such existence, see `Mizar_struct` file). For example, for `doubleLoopStr` we use:

```
term "{⟨carrier, the set⟩} ∪
  {⟨addF, the BinOp-of the set⟩} ∪ {⟨ZeroF, the Element-of the set⟩} ∪
  {⟨multF, the BinOp-of the set⟩} ∪ {⟨OneF,the Element-of the set⟩}"
```

3.4 Inheritance and Multiple Inheritance

The original MML definition of `doubleLoopStr` includes information about (multiple) inheritance:

```
struct (addLoopStr,multLoopStr_0) doubleLoopStr
```

which informs Mizar that `doubleLoopStr` should inherit all the fields contained in `addLoopStr`, as well as those in `multLoopStr_0`. These are used to define additive and multiplicative groups, respectively. Inheritance is transitive. Multiple inheritance causes the Mizar inheritance graph to become a DAG. There are 168 structures defined in MML. This does not include their versions with adjectives. Most structures (135 of them) inherit from `1-sorted`. A part of the graph restricted to the most basic algebraic structures is depicted in Fig. 1. In our approach it is possible to verify that the domain of a structure is a subset of the domain of another one allowing (automated) inheritance proofs at any point after the definition.

4 Set Comprehension

Set comprehension is a key notion in Mizar set theory. It allows defining a set of terms, which satisfy the given predicate (see [6, Fraenkel]), with Mizar syntax:

$$\{\mathtt{t}(\mathtt{v}_1,\mathtt{v}_2,\ldots,\mathtt{v}_n) \text{ where } \mathtt{v}_1 \text{ is } \Theta_1,\ \mathtt{v}_2 \text{ is } \Theta_2,\ldots,\mathtt{v}_n \text{ is } \Theta_n{:}\mathtt{P}[\mathtt{v}_1,\mathtt{v}_2,\ldots,\mathtt{v}_n]\}$$

Such an expression is of the type `set` in Mizar and it is not possible to further specify the type. The built-in definition of the set comprehension operator is automatically expanded in terms of set membership as follows:

$$\mathtt{x} \text{ in } \{\mathtt{t}(\mathtt{v}_1,\mathtt{v}_2,\ldots,\mathtt{v}_n) \text{ where } \mathtt{v}_1 \text{ is } \Theta_1,\ \mathtt{v}_2 \text{ is } \Theta_2,\ldots,\mathtt{v}_n \text{ is } \Theta_n : \mathtt{P}(\mathtt{v}_1,\mathtt{v}_2,\ldots,\mathtt{v}_n)\}$$
$$\text{iff}$$
$$\text{ex } \mathtt{v}_1 \text{ be } \Theta_1,\ \mathtt{v}_2 \text{ be } \Theta_2,\ \ldots,\mathtt{v}_n \text{ be } \Theta_n \text{ st } \mathtt{x} = \mathtt{t}(\mathtt{v}_1,\mathtt{v}_2,\ldots,\mathtt{v}_n)\ \&\ \mathtt{P}[\mathtt{v}_1,\mathtt{v}_2,\ldots,\mathtt{v}_n]$$

The generality of the definition could quickly lead to a version of the Russell's paradox, as according to the Tarski-Groethendieck axiomatization everything is a set. Therefore, the set comprehension operator is well-formed only when all the types $\Theta_1, \Theta_2, \ldots, \Theta_n$ have the sethood property (otherwise Mizar reports Error 86: *"It is only meaningful for sethood property"*, see [7] for more details).

Definition 1. *A Mizar-type Θ has the sethood property if all objects of the type Θ are elements of some set.*

We define sethood in Isabelle/Mizar and make sure that it is proved for the most important Mizar types (Mizar allows the inheritance of sethood). With this, we can show the existence of sets described by comprehensions. The Isabelle/Mizar statement and proof are quite involved, so we present these mostly in mathematical setting.

Theorem 1. *Let Θ be a Mizar type with the sethood property, P be a unary predicate and F be a unary function defined on Θ. Then there exists a set C such that each x is a member of C if and only if there exists a v of type Θ such that $x = F(v) \wedge P(v)$.*

Proof. The proof only relies on the Tarski-Groethendieck axiom of Replacement. Consider the set S_{ethood} that contains all objects of the type Θ. Furthermore, consider the binary relation R_1 defined for a predicate P as $R_1(x, y) \iff x = y \wedge P(x)$. Then, by the axiom of Replacement, there exists a set $S_{eparation}$, such that x is a member of $S_{eparation}$ if and only if there exists y that is a member of S_{ethood} and $R_1(x, y)$. Now $S_{eparation}$ contains the objects of the type Θ which satisfy P and only such objects. We can use the Replacement axiom again for the unary relation $\lambda y.\exists x.\, y = F(x)$ and the set S_{ethood}. This gives the image of the function F on the set S_{ethood}. This set fulfills the requirement of the theorem statement. □

The theorem was so far limited to unary predicates and functions. To adapt it to multiple arguments, we can consider the Cartesian product together with the property that two tuples are equal, if their corresponding elements are equal. In our Isabelle/Mizar formalization we introduced the Cartesian product in the zfmisc_1 theory corresponding to the Mizar article with the same name. This can used to show set comprehensions with multiple arguments:

Theorem 2. *Let $\Theta_1, \Theta_2, \ldots, \Theta_n$ be Mizar types with the* sethood *property, P be an n-argument predicate and F be an n-argument function defined for the arguments of the types $\Theta_1, \Theta_2, \ldots, \Theta_n$. Then there exists a set C such that x is a member of C if and only if there exists v_1 be Θ_1, v_2 be Θ_2, ..., v_n be Θ_n, such that $x = F(v_1, v_2, \ldots, v_n) \wedge P(v_1, v_2, \ldots, v_n)$.*

Proof. Consider the sets S_i which contain objects of the types Θ_i. Consider the binary relation R_1, defined as

$$\lambda xy.x = y \wedge \exists_{v_1,v_2,\ldots,v_n} x = \langle\langle \ldots \langle\langle v_1, v_2\rangle, v_3\rangle, \ldots\rangle, v_n\rangle \wedge P(v_1, v_2, \ldots, v_n)$$

The Replacement axiom can be used to obtain the set $S_{eparation}$ for which x is a member of $S_{eparation}$ if and only if there exists y that is a member of $(\ldots((S_1 \times S_2) \times S_3)\ldots) \times S_n$ and $R_1(x, y)$. Using the Replacement axiom again for the relation

$$\lambda xy.\exists_{v_1,v_2,\ldots,v_n} x = \langle\langle \ldots \langle\langle v_1, v_2\rangle, v_3\rangle, \ldots\rangle, v_n\rangle \wedge y = F(v_1, v_2, \ldots, v_n)$$

and the set S_{ethood}, we obtain the set C. □

The following example collects the results of the function `f` on the set `X` and can be shown to be equivalent to the range of the function restricted to the set.

```
term "{f. x where x be Element-of dom f: x in X}"
```

Set comprehensions are often used in the MML to define sets of terms without additional properties. The following syntax has been introduced so simplify such comprehension terms: the set of all $\mathtt{t}(\mathtt{v}_1,\mathtt{v}_2,\ldots,\mathtt{v}_n)$ where $\mathtt{v}_1$ is Θ_1, $\mathtt{v}_2$ is Θ_2,...,$\mathtt{v}_n$ is Θ_n which abbreviates: {$\mathtt{t}(\mathtt{v}_1,\mathtt{v}_2,\ldots,\mathtt{v}_n)$ where $\mathtt{v}_1$ is Θ_1, $\mathtt{v}_2$ is Θ_2,...,$\mathtt{v}_n$ is Θ_2: non contradiction}. Just like for set comprehensions we add this abbreviation together with the Mizar notation. It can be seen for example in the following theorem:

```
theorem funct_1_th_110:
  assumes "B be non empty | functional ‖ set"
          "f be Function" "f = union B"
  shows
     "dom f = union the set-of-all dom g where g be Element-of B"
     "rng f = union the set-of-all rng g where g be Element-of B"
```

5 Case Studies

In this section we argue that our model of structures is not only correct based on the Tarski-Groethendieck set theory axioms, but also that it is adequate for Mizar-like formalization. For this, we formalized a part of Mizar's group theory in Isabelle defining the basic concepts as structures, the corresponding attributes, and showing a number of their properties. We also show how groups combine with set comprehensions and a more involved inheritance example. All Isabelle examples have same identifiers as their Mizar counterparts to ease comparison.

5.1 Algebraic structures

We first define the Mizar type of groups as the multiplicative magma structure `multMagma` with three adjectives. We define the identity in the group and an inverse, where the group operation is defined as usual.

```
abbreviation Group where
   "Group ≡ Group-like | associative | non empty-struct ‖ multMagma"

definition group_1_def_4 ("1'._" [1000] 99) where
  "assume G is unital
   func 1.G → Element-of-struct G means λit.
      for h being Element-of-struct G holds
         h ⊗G it = h & it ⊗G h = h"

definition group_1_def_5 (infix "⁻¹" 105) where
  "func h⁻¹G → Element-of-struct G means λit.
      h ⊗G it = 1.G & it ⊗G h = 1.G"

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 ⦈"
```

Next, we show a number of theorems about groups. We show here only a property that each group fulfils properties of semigroups with involution. The Mizar formalization does not need to repeat the variable declarations, thanks to the reserve mechanism, which is similar to Isabelle locales, but for each theorem only the variables and assumptions that are actually needed for its statement are exported. We do not have a complete mechanism of this kind yet.

```
theorem group_1_th_16:
  assumes "G be Group"
          "h be Element-of-struct G" "g be Element-of-struct G"
  shows "(h ⊗G g)⁻¹G = g⁻¹G ⊗G h⁻¹G"
```

We have also reproved the 13 schemes that talk about set comprehension. We show here two, one that combines set comprehension with functions, and one that uses nested comprehensions.

```
theorem Fraenkel_sch_9:
  assumes "A be set" "B be set" "X be set"
          "f be Function-of A,B" "g be Function-of A,B"
          "(f | X) = (g | X)"
        "for u being Element-of A st u in X holds P(u) iff Q(u)"
  shows "{ f. u where u be Element-of A : P(u) & u in X } =
         { g. v where v be Element-of A : Q(v) & v in X }"

theorem Fraenkel_sch_13:
  assumes T0: "A be set" "B be set" "C be set"
   "for x1 be object,x2 be object holds F(x1,x2) be Element-of C"
  shows "{ st1 where st1 be Element-of C:
              st1 in {F(s1,t1) where s1 be Element-of A,
                 t1 be Element-of B: P(s1,t1) } & Q(st1)} =
         { F(s2,t2) where s2 be Element-of A,t2 be Element-of B:
              P(s2,t2) & Q(F(s2,t2))}"
```

We finally look at the combination of groups and set comprehensions. The following two definitions introduce the set of all inverses and the set of results of the group operation:

```
definition group_2_def_1 (infix "˜⁻¹ » 150) where
  "func A˜⁻¹G → Subset-of-struct G equals
     {g⁻¹G where g be Element-of-struct G : g in A}"

definition group_2_def_2(" _  ⊗_ _" [66, 1000, 67] 66) where
  "func A  ⊗G B → Subset-of-struct G equals
     {a ⊗G b where a be Element-of-struct G,
                   b be Element-of-struct G : a in A & b in B}"
```

We can now show the relationship between these two operations, which is a consequence of the properties of semigroups with involution (`group_1_th_16` above).

```
theorem group_2_th_11:
  assumes "G be Group"
          "A be Subset-of-struct G" "B be Subset-of-struct G"
  shows "(A  ⊗G B) ˜⁻¹G = B˜⁻¹G  ⊗G A˜⁻¹G"
```

We finally show a multiple inheritance relation for the double loop structure. It follows by simple rewriting just using the definitions of the structures.

```
theorem doubleLoopStr_inheritance:
  assumes "X be doubleLoopStr"
  shows "X be multLoopStr_0" "X be addLoopStr"
```

5.2 SCM computer model

The MML models computers as structures whose elements correspond to: the set of instructions, the computer memory, and the functor `Execution` whose role is to map each instruction to a function from memory states to memory states.

The instructions form a set which must fulfill four properties corresponding to the following adjectives

```
abbreviation
  "Instructions ≡ J/A-independent|homogeneous|with_halt|standard-ins||set"
```

The standard-ins adjective specifies that any element i of the instruction set is a triple

```
term "[InsCode i, JumpPart i, AddressPart i]"
```

where InsCode i is an instruction number represented by a natural number, JumpPart i is a list of natural numbers used by the Execution functor to compute the following instruction numbers, and AddressPart i is a list objects passed to the instructions as arguments. The with_halt adjective means that the set I includes a halt instruction. The halt instruction is represented as [0,{},{}], where the empty set corresponds to the empty list. The adjectives homogeneous and J|A-independent specify a subset of instructions which share the number InsCode and are necessary for the definition of the Execution functor. homogeneous specifies, that the JumpPart lists of arguments given to the InsCode instruction are always of the same length (for example goto always requires one argument). J|A-independent specifies that every list of the appropriate length can be handled (for the goto example, Execution must be able to perform a goto instruction to every location).

```
definition compos_0_def_5 ("homogeneous") where
  "attr homogeneous means (λI.
     I be non empty|standard-ins||set &
      (for i,j be Element-of I st InsCode i = InsCode j holds
         dom JumpPart i = dom JumpPart j))"

definition compos_0_def_7 ("J/A-independent") where
  "attr J/A-independent means (λI.
     I be non empty|standard-ins||set &
     (for n be Nat, f1,f2 be NAT-valued ||Function, p be object
        st dom f1 = dom f2 & [n,f1,p] in I holds [n,f2,p] in I))"
```

The next structure in the formalization is the computer memory (see Fig. 1). It is also modeled as a structure. The main field, carrier, corresponds to the actual memory and the set N gives the kind of data that can be stored within it. Note that that in SCMs all memory locations are of the same size [17]. The ZeroF field is the instruction counter. It corresponds to the number of the instruction performed in the given state. Object-Kind indicates the kind of data stored in the given memory location and ValuesF gives the value range for the given type.

```
definition MemStruct_over ("Mem-Struct-over _") where
"struct Mem-Struct-over N (#
   carrier → λS. set;
   ZeroF → λS. Element-of the carrier of S;
   Object-Kind → λS. Function-of the carrier of S, N;
   ValuesF →  λS. ManySortedSet-of N
#)"
```

An actual memory state is defined as a function that associates each memory location in the carrier with the stored data, where the value must be one of the allowed values.

```
definition memstr_0_def_2 ( "the'_Values'_of _ " 190) where
  "func the_Values_of M → ManySortedSet-of the carrier of M equals
     the ValuesF of M ∘ the Object-Kind of M"

abbreviation memstr_0_mode_2 ("State-of _" 190)
  where "State-of M ≡
         (the carrier of M):total | the_Values_of M-compatible ‖ Function"
```

We can now formulate the Mizar type of a computer and show the non-emptiness of this type. The type is a structure parametrized by the data stored in the memory of the computer. The name AMI (for *Architecture Model for Instructions*) is used in the Mizar dictionaries to refer to this structure as a type and SCM is an object of the type [17].

```
definition AMI_Struct_over ("AMI-Struct-over _") where
  "struct AMI-Struct-over N (#
     carrier → λS. set;
     ZeroF → λS. Element-of the carrier of S;
     InstructionsF → λ_. Instructions;
     Object-Kind → λS. Function-of the carrier of S, N;
     ValuesF →  λS. ManySortedSet-of N;
     Execution → λS. Action-of the InstructionsF of S,
          product ((the ValuesF of S)*‘the Object-Kind of S)#)"
```

We subsequently reformalize a machine with the halt instruction and we show that all the indicated fields have their corresponding types, and that this construction uniquely defines a computer. The proof corresponding to the below definition requires 83 lines of Isabelle proof to justify.

```
definition extpro_1_def_1 ("Trivial-AMI _") where
  "func Trivial-AMI N → strict AMI-Struct-over N‖AMI-Struct-over N
     means (λit.
      the carrier of it = {0} &
        the ZeroF of it = 0 &
the InstructionsF of it = {[0,{},{}]} &
  the Object-Kind of it = {0} --> 0  &
      the ValuesF of it = N --> NAT &
    the Execution of it = {[0,{},{}]} --> id product(N --> NAT ∘ {0} --> 0))"
```

Next, we introduce the `Exec` functor. Applying the instruction `I` to `(the Execution of S)` we should obtain a function that can be given a memory state as input and returns a memory state. Again showing the correctness of the definitions and that these properties hold requires 57 lines of Isabelle proofs.

```
definition extpro_1_def_2( "Exec _'(_ ,  _')" 190) where
  "func Exec S(I,s) → State-of S equals
    ((the Execution of S).I).s "

definition extpro_1_def_3("halting _") where
  "attr halting S means (λI.
       I be Instruction-of S &
       (for s be State-of S holds  Exec S(I,s) = s))"
```

We finally show, that `Trivial-AMI N` is of the computer type and that it does halt, which shows the non-emptiness of the Mizar type of computers.

```
theorem extpro_1:
  assumes "N be with_zero‖set"
  shows "halt Trivial-AMI N is halting Trivial-AMI N"
```

6 Conclusion

Mizar structures and set comprehension operators complete the foundations of Mizar as an Isabelle object logic. This allows manual translation of the MML to Isabelle/Mizar, as we have shown with the Mizar theory of basic algebraic structures including SCMs. We have defined 90 concepts where 27 of them required justifications and proved 105 registrations, 31 theorems that discuss based algebraic structures and set comprehensions, as well as inheritance relations between 15 structures. We have defined also 27 concepts where 14 of them required justification and proved 12 registrations, 3 theorems about SCMs. The total combined size of the development is 513 kB and 8295 lines of proofs. It is available at:

`http://cl-informatik.uibk.ac.at/cek/macis2017/`

The Isabelle proofs are mostly longer are than their Mizar counterparts. This is predominantly because of the lack of type automation for the type system, even if the Mizar type system could be handled by ATPs [15]. Similarly, many Isabelle proofs require more labels than the corresponding Mizar ones, which we hope to remedy by developing legibility tools similar to the ones available for Mizar [22]. Finally it would be interesting to mechanically translate MML statements or even proofs and imitate the behavior of Mizar's automation.

Acknowledgements This work has been supported by the European Research Council (ERC) grant no. 714034 *SMART* and the Polish National Science Center granted by decision n°DEC-2015/19/D/ST6/01473.

References

1. Abrial, J.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010)
2. Asperti, A., Ricciotti, W.: A Formalization of Multi-tape Turing Machines. Theor. Comput. Sci. 603, 23–42 (2015)
3. Brown, C.E., Urban, J.: Extracting Higher-order Goals from the Mizar Mathematical Library. In: Kohlhase, M., Johansson, M., Miller, B.R., de Moura, L., Tompa, F.W. (eds.) Proc. 9th International Conference on Intelligent Computer Mathematics (CICM 2016). LNCS, vol. 9791, pp. 99–114. Springer (2016)
4. Bulwahn, L., Krauss, A., Haftmann, F., Erkök, L., Matthews, J.: Imperative Functional Programming with Isabelle/HOL. In: Mohamed, O.A., Muñoz, C.A., Tahar, S. (eds.) Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008. LNCS, vol. 5170, pp. 134–149. Springer (2008)
5. Geuvers, H., Pollack, R., Wiedijk, F., Zwanenburg, J.: A Constructive Algebraic Hierarchy in Coq. J. Symb. Comput. 34(4), 271–286 (2002)
6. Grabowski, A., Korniłowicz, A., Naumowicz, A.: Mizar in a Nutshell. J. Formalized Reasoning 3(2), 153–245 (2010)
7. Grabowski, A., Korniłowicz, A., Naumowicz, A.: Four Decades of Mizar. Journal of Automated Reasoning 55(3), 191–198 (2015)
8. Grabowski, A., Korniłowicz, A., Schwarzweller, C.: On Algebraic Hierarchies in Mathematical Repository of Mizar. In: Ganzha, M., Maciaszek, L.A., Paprzycki, M. (eds.) Proc. Federated Conference on Computer Science and Information Systems (FedCSIS 2016). pp. 363–371 (2016)

9. Haftmann, F., Wenzel, M.: Constructive Type Classes in Isabelle. In: Altenkirch, T., McBride, C. (eds.) Types for Proofs and Programs, International Workshop, TYPES 2006. LNCS, vol. 4502, pp. 160–174. Springer (2007)
10. Harrison, J., Urban, J., Wiedijk, F.: History of Interactive Theorem Proving. In: Siekmann, J.H. (ed.) Computational Logic, Handbook of the History of Logic, vol. 9, pp. 135–214. Elsevier (2014)
11. Iancu, M., Kohlhase, M., Rabe, F., Urban, J.: The Mizar Mathematical Library in OMDoc: Translation and Applications. J. Autom. Reasoning 50(2), 191–202 (2013)
12. Kaliszyk, C., Pąk, K.: Presentation and Manipulation of Mizar Properties in an Isabelle Object Logic. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) 10th International Conference on Intelligent Computer Mathematics (CICM'17). LNCS, vol. 10383, pp. 193–207. Springer (2017)
13. Kaliszyk, C., Pąk, K., Urban, J.: Towards a Mizar Environment for Isabelle: Foundations and Language. In: Avigad, J., Chlipala, A. (eds.) Proc. of 5th Conference on Certified Programs and Proofs (CPP 2016). pp. 58–65. ACM (2016)
14. Kaliszyk, C., Pąk, K.: Progress in the Independent Certification of Mizar Mathematical Library in Isabelle. In: Ganzha, M., Maciaszek, L.A., Paprzycki, M. (eds.) Proc. Federated Conference on Computer Science and Information Systems (FedCSIS 2017), pp. 227–236 (2017)
15. Kaliszyk, C., Urban, J.: MizAR 40 for Mizar 40. J. Autom. Reasoning 55(3), 245–256 (2015)
16. Kaliszyk, C., Wiedijk, F.: Merging Procedural and Declarative Proof. In: Berardi, S., Damiani, F., de'Liguoro, U. (eds.) Types for Proofs and Programs, International Conference, TYPES 2008. LNCS, vol. 5497, pp. 203–219. Springer (2008)
17. Korniłowicz, A., Schwarzweller, C.: Computers and Algorithms in Mizar. Mechanized Mathematics and Its Applications 4(1), 43–50 (2005)
18. Lammich, P.: Refinement to Imperative/HOL. In: Urban, C., Zhang, X. (eds.) Interactive Theorem Proving - 6th International Conference, ITP 2015. LNCS, vol. 9236, pp. 253–269. Springer (2015)
19. Lee, G., Rudnicki, P.: Alternative Aggregates in Mizar. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) Mathematical Knowledge Management (MKM 2007). LNCS, vol. 4573, pp. 327–341. Springer (2007)
20. Megill, N.D.: Metamath: A Computer Language for Pure Mathematics. Lulu Press, Morrisville, North Carolina (2007)
21. Nakamura, Y., Trybulec, A.: A Mathematical Model of CPU. Formalized Mathematics 3(**2**), 151–160 (1992)
22. Pąk, K.: Automated Improving of Proof Legibility in the Mizar System. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) Intelligent Computer Mathematics, CICM 2014. LNCS, vol. 8543, pp. 373–387. Springer (2014)
23. Sacerdoti-Coen, C., Tassi, E.: Formalising Overlap Algebras in Matita. Mathematical Structures in Computer Science 21(4), 763–793 (2011)
24. Wenzel, M., Paulson, L.C., Nipkow, T.: The Isabelle Framework. In: Mohamed, O.A., Muñoz, C.A., Tahar, S. (eds.) Theorem Proving in Higher Order Logics, TPHOLs 2008. LNCS, vol. 5170, pp. 33–38. Springer (2008)
25. Wiedijk, F.: Mizar's Soft Type System. In: Schneider, K., Brandt, J. (eds.) Proc. Theorem Proving in Higher Order Logics, 20th International Conference, TPHOLs 2007, LNCS, vol. 4732, pp. 383–399. Springer (2007)
26. Xu, J., Zhang, X., Urban, C.: Mechanising Turing Machines and Computability Theory in Isabelle/HOL. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) Interactive Theorem Proving - 4th International Conference, ITP 2013. LNCS, vol. 7998, pp. 147–162. Springer (2013)