Presentation and Manipulation of Mizar Properties in an Isabelle Object Logic*

Cezary Kaliszyk[1] and Karol Pąk[2]

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

Abstract. One of the crucial factors enabling an efficient use of a logical framework is the convenience of entering, manipulating, and presenting object logic constants, statements, and proofs. In this paper, we discuss various elements of the Mizar language and the possible ways how these can be represented in the Isabelle framework in order to allow a suitable way of working in typed set theory. We explain the interpretation of various components declared in each Mizar article environment and create Isabelle attributes and outer syntax that allow simulating them. We further discuss introducing notations for symbols defined in the Mizar Mathematical Library, but also synonyms and redefinitions of such symbols. We also compare the language elements corresponding to the actual proofs, with special care for implicit proof expansions not present in Isabelle. We finally discuss Mizar's hidden arguments and demonstrate that some of them are not necessary in an Isabelle representation.

1 Introduction

Set theory has been the standard foundation used by mathematicians in the past and is still the foundation with which the majority are most familiar today. However, when it comes to formalized mathematics, most proof assistants are based on other foundations. This could be one of the important factors that are hindering a wider adoption of formal proof. Mizar [4] has been the pioneering proof assistant based on classical logic combined with the axioms of set theory. It also offers several unique features commonly used by mathematicians, such as powerful symbol overloading, e.g. the plus symbol can be used to represent more than one hundred different kinds of addition that are recognized based on the types of arguments. Additionally, several kinds of additions can be used simultaneously if arguments of more specific types can be applied to different plus that give the same result for such types. It also features one of the largest libraries of formalized mathematics, the Mizar Mathematical Library (MML) [2]. Unfortunately the complete foundations of Mizar have only been specified in documentation and by the algorithm implemented by the Mizar system, rather than in a fully formal way.

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

Furthermore the implementation of Mizar does not have a small core, to which all the proofs could be reduced, as it is in the case of the implementations of many proof assistants [20]. To overcome these limitations, in our previous work [6] we have proposed an Isabelle [19] object logic [13], which implements the core foundations of Mizar. We have expressed most features supported by Mizar logic including the Mizar type system, basic mechanisms for introducing definitions, and we proved the first 32 facts from the MML including theorems, registrations as well as definitions that required Mizar-style justifications. The main motivations for the project are to provide an independent certification of the MML and to provide easy access to MML data from other systems. This can in the long run give an alternative Mizar implementation.

One problem with the emulation of Mizar as an Isabelle object logic was the convenience of entering and manipulating the Mizar objects, statements, and proofs. Many properties that in Mizar could be introduced using concise and convenient notations (such as for example Mizar clusters) would require explicitly specifying existential statements or even meta-logic implications.

In this paper we investigate the features of Mizar which allow convenient input and manipulation of the set theoretic objects and explore the possibilities to provide similar mechanisms in an Isabelle object logic. The main motivation for this particular work is to give notations usable first in the reference implementation of Mizar foundations, with further work allowing to translate the whole MML to the proposed syntax in an automated way.

1.1 Related Work

Many packages and improvements have been introduced in the Isabelle/HOL object logic that allow convenient defining, manipulating, and display of its objects. Default type-classes are assumed, `Trueprop`[3] is automatically added around HOL propositions, packages allow various ways of introducing definitions and type definitions conveniently. Isabelle and many other logical frameworks (Twelf [17], MMT [16]) allow adding nice notations to object-logic defined functions and other entities.

Some mechanisms of this kind have been also used and developed for Isabelle/ZF [14]. Further improvements of some of its features have been also considered in other systems that implement set theory. ProofPeer [11] allows using the same notation for meta-level application and ZF application and is able to infer the kind of application at each instance [12]. The Atelier B including Rodin [1], as well as Metamath [10] include specific user interface techniques for nicer set theoretic notations. Furthermore, both specific tactic and various general purpose automation techniques have been considered to provide type inference in various Isabelle object logics. Mizar type inference can be to a large extent automated using first-order automated theorem provers [7,18].

[3] `Trueprop` is the HOL object logic constant that turns a higher-order logic boolean into a meta-level proposition.

There has been a lot of work on improving legibility of proofs, in particular for the Mizar proof language and proof style [9]. The main purpose has been to present the proofs in a concise but readable way [15], whereas in this paper we are trying to focus on the outer and inner syntax for Isabelle/Mizar rather than on the proof structure.

Bancerek [3] has developed a formal theory of Mizar linguistic concepts, which includes quasi-loci, quasi-terms, quasi-adjectives, and quasi-types, which is accompanied by the Mizar article `ABCMIZ_0` formalizing the Mizar type system.

1.2 Paper Content and Contributions

After shortly recalling the basics of our Isabelle/Mizar object logic in Section 2 and explaning some improvements in its foundations, we propose the mechanisms that allow imitating the notations and the presentation of Mizar objects. In particular:

- We propose a syntax for Mizar-like definitions of all kinds of functors, modes, predicates, and attributes. This includes support for conditional definitions, where the given types are not sufficient to guarantee the soundness of the definitions (Sec. 3).
- We discuss the background information available in Mizar and propose mechanisms that allow extending this information in Isabelle (Sec. 4), in particular environment declarations, properties, and registrations.
- We propose a mechanism that allows hiding certain functor arguments in the Isabelle object logic, that mimics the hidden argument inference present in Mizar. The proposed mechanism allows hiding the arguments that appear neither in the definition body nor in the result type. Moreover, we introduce Mizar-style redefinitions and propose how arguments that do not appear in original definitions can be hidden (Sec. 5).

2 Mizar Object Logic

In this section we shortly remind the basics of the Mizar object logic fully introduced by us in [6] and present the new developments in this foundation.

We start with the Isabelle/FOL object logic. We disable the notations $=, \forall, \exists$ of the first-order polymorphic equality, and polymorphic quantifiers. We instead introduce the Isabelle types for Mizar sets, Mizar types, and type modifiers and restrict the symbols for equality and quantifiers to the type of Mizar sets. We further introduce new constants with appropriate axiomatizations that allow the introduction of concrete Mizar types and their modifiers, as well as mechanisms that allow establishing such definitions given the proofs of the necessary conditions (definitions are further discussed in the next Section 3). Given these foundations, the same axiomatization of set theory as in Mizar (Tarski-Grothendieck) can next be introduced, and can be followed by the same definitions to build up a standard library of set theory.

We have introduced some simplifcations and optimizations in the definition of the foundations since it was proposed [6]. We separated types from type modifiers, with the aim of automating some type inference. We were able to remove the axiom that postulated the existence of functors, if their existence and uniqueness were shown. Instead, the definition body can be converted into a temporary attribute (type modifier), from which global choice can give the attributes. Finally, we separated the type membership operator be from the type modifier membership is, to reduce confusion. We will discuss changes in the definition mechanism in the next Section.

3 Definition Syntax

Many logical foundations include not only inference rules, but also rules that allow extending the actual logic, such as introducing definitions and type definitions in higher-order logic. For a logical framework to fully emulate such logics, it is necessary to adequately enable such extension mechanisms. In our previous work [6] we showed that a number of basic Mizar definitions can be directly emulated in Isabelle. However, we need to introduce a more involved definition mechanism to cover the more advanced Mizar definitions including conditional ones (called *permissive* in Mizar), as well as the definitions of new modes and attributes, which are the focus of this section.

All Mizar logic extensions are introduced in *definition* blocks. A definition block can be made up of (multiple) declarations of the types of definition arguments (referred to as *loci declarations* in Mizar), optional assumptions, and the actual definitions which can use these declarations. The Mizar CHECKER verifies that the argument types employed in the definition block are non-empty. It is possible to apply types for which non-emptiness has not been proved using the optional assumptions. The actual defined object can be of four kinds: meta-level functions and constants (referred to as *functors* in Mizar), Mizar soft types (referred to as *modes* in Mizar), predicates, and type restrictions (*attributes* in Mizar). In all these cases the defined object together with its its list of arguments constitutes the definition pattern, where the arguments are the bound variables introduced inside the current definitional block whose types are non-empty. Note, that not all arguments must be visible in the pattern, hidden arguments will be discussed in Sec. 5.

3.1 Conditional Definitions

Definitions with assumptions explicitly stated in the definition block are very frequent in Mizar [5]. Such definitions are referred to as *permissive definitions* in Mizar literature. The assumptions are often used to express the relations between the arguments in cases where the type (non-empty ones) and attributes is not powerful enough to capture all the definedness conditions. Consider the conditional assumption "polynomial has a root". It can be used to define the type

of a root of an arbitrary non-zero degree polynomial over an arbitrary ring (not restricted to algebraically closed fields).

The majority of the conditional definitions contained in the MML are not however there to overcome the non-emptiness problem. Their main role is to provide more freedom in the formulation of reasoning steps, in such a way that they are verified by the Mizar system. This is desired, because of how Mizar identifies which defined object does the current symbol refer to: as the patterns used in the definitions can be overshadowed by subsequent definitions, the Mizar ANALYZER must consider the types of the objects in the argument list. These types are computed based on the explicitly given types of argument variables together with the information contained in the clusters (Section 4.2, [4,5]).

Conditional definitions can therefore be used to allow writing terms, which can be correctly identified even given more general argument types, but whose interpretation is only possible with more precise types given in the definition assumptions. Consider the conditional definition of projection introduced in `XTUPLE_0` (left) together with its non-conditional variant (right), as well as two formulas that use this functor (we will arrive at this definition in Isabelle/Mizar at the end of this section):

```
definition
  let x be object;
  assume x is pair;
  func x`1 -> object means
    x = [y1,y2] implies it = y1;
end;
```

```
definition
  let x be pair object;

  func x`1 -> object means
    x = [y1,y2] implies it = y1;
end;
```

```
for x be pair object holds x`1 = ...
for x be object st x is pair holds x`1 = ...
```

The conditional definition allows the projection functor in the term `x`1` to be correctly identified by Mizar in both formulas. However, in the case of the non-conditional variant, the ANALYZER cannot identify the term: Given the type of `x` explicitly given as `object`, the functor `` `1 `` with the argument `x` cannot be substituted in the definition and the unknown functor 103 error is reported.

3.2 Functor Definitions

The functions in Mizar's first-order logic (from the logical framework point of view these are meta-level functions) are referred to as *functors* in Mizar. New functors are defined in a definition block using the keyword func followed by the functor pattern (for example `f.x` for the image of `x` under function `f`), the result type (which must be non-empty) and the actual definition body. There are two types of functor definitions that differ in the form of the body of the definition: definitions by equals and by means.

A definition by equals equates the functor given the arguments to a Mizar term, with the type constraint. Definitions by means allow the define functors indirectly, by giving the intended properties. The properties can use all the

parameters introduced in the current definitional block, together with a special variable it, which marks the defined functor. The following abbreviations allows for Mizar-like syntax for introducing the definition in Isabelle, further using λit to bind the currently defined object:

```
abbreviation (input) equals_prefix
   ("func _ → _ equals _" [10,10] 10)
where "func pat → type equals def ≡
   pat = theM (λit. it be type & it = def)"

abbreviation (input) means_prefix
   ("func _ → _ means _" [0,0] 10)
where " func pat → type means condition ≡
   pat = theM (λit. (it be type & condition(it)))"
```

Using a definition by means forces the user to prove two correctness conditions (proofs required by the user to ensure the soundness of a definition), existence and uniqueness immediately in the definitional block, i.e., there exists an object that has the result type and fulfills the property, as well as any two objects of the type which satisfy the property must be equal. Mizar uses these correctness conditions to create a definitional theorem from the definition body, which we can formalize once in Isabelle:

```
lemma means_property:
assumes df: "f = theM(λx. x be D & P(x))"
    and ex: "ex x being D st P (x)"
    and un: "⋀x y. x be D ⟹ y be D ⟹
         P (x) ⟹ P (y) ⟹ x = y"
 shows "f be D & P(f) & (x be D & P(x) implies x = f)"
```

A definition by equals can be considered as a special case of a definition by means, where the definition body has the form it = ..., where it does not occur on the right-hand side. The correctness conditions can be simplified to the coherence condition, i.e that the term on the right-hand side has the correct result type. The definitional theorem is also simplified:

```
lemma equals_property:
assumes df: "f = theM(λx. x be D & x=g)"
   and coherence: "g be D"
 shows "f be D & f = g"
```

Mizar allows conditional definitions for both definitions by equals and by means. The presence of assumptions in the definition block implies more complex definitional theorems, which include properties of functor definitions both when the variables satisfy the assumptions and when it is not the case. If the assumptions are not satisfied, the only information the Mizar CHECKER has about the term is its result type.

To obtain the same behavior in Isabelle, we introduce the following notation:

```
abbreviation(input) assume_means_prefix
   ("assume _ func _ → _ means _" [0,0,0,0] 10)
where "assume as func def → type means condition ≡
  def = theM (λit.(it be type & (as implies condition (it))))"
```

together with an interpretation of the definitional theorem, where $q : Q$ is used for the assumptions:

```
lemma assume_means_property:
assumes df: "f = theM(λx. (x be D & (R implies P (x))))"
   and q: Q
   and assume_ex: "R ⟹ ex x being D st P(x)"
   and assume_un: "⋀x y. R ⟹ x be D ⟹ y be D ⟹
       P (x) ⟹ P (y) ⟹ x = y"
   and mode_ex: "ex x being  D st True"
shows
   "f be D & (R implies P(f) & (x be D & (P (x)) implies x = f))"
```

The conclusion of the theorem corresponds exactly to the information exposed by Mizar to the user. The proof of the theorem proceeds in different ways depending on whether the assumptions are true: If `R` holds, lemma *means_property* can be immediately used, if it does not, it is only necessary to show the type of the definition which follows from the description operator. The indefinite description operator will be denoted in this paper as theM.

We can finally introduce the conditional definition of projection in Isabelle, that was the motivation for this section, with a syntax that is as convenient as that given by Mizar, which given the same as the Mizar proofs of existence and uniqueness is equivalent to the original one.

```
definition xtuple_0_def_2_prefix (" _ `1" [90] 95) where
  "assume x is pair
   func x `1 → object means
     (λit. for y1,y2 be object st x=[y1,y2] holds it = y1)"
```

3.3 Expandable Modes

The Mizar type system can be extended with new types (referred to as mode in Mizar) in two ways. First, *expandable modes* allow the user to introduce an abbreviation for any Mizar type that can already be expressed as a collection of attributes applied to an already defined radix type (type without attributes). Second, *non-expandable modes* allow the user to introduce a type which is a subtype of an existing type, which additionally satisfies the condition given in the definition body. Just like in the case of defining functors, in both mode cases the pattern occurs with a fixed list of arguments introduced in the current definitional block. For non-expandable modes, additionally the parent type and the definition body must be given, and again it can be used to refer to the mode being defined.

In order to use the defined mode in Mizar, it is necessary to show the existence condition, i.e. that there exist at least one element of the defined type. In case of non-expandable modes, this must be shown in the definition block, whereas for *expandable modes*, Mizar attempts to automatically show it using the defined clusters available in the article's background knowledge (the details on the treatment of background knowledge will be discussed in Section 4), and if it is not possible reports the 136 *Non registered cluster* error.

Expandable modes are considered standard abbreviations internally expanded by Mizar into a collection of adjectives associated with a radix type. This expansion is repeated until the radix type in a non-expandable mode. Therefore, we introduce such types as abbreviations in the Isabelle emulation.

```
abbreviation funct_2_def_1 ("Function-of _ , _" 190)
where "Function-of X,Y ≡ (X,Y: quasi-total) ‖ (PartFunc-of X,Y)"
```

3.4 Non-Expandable Mode and Attribute Definitions

For non-expandable modes we again introduce two notations that allow introducing new types using definitions without assumptions, as well as conditional ones. Together with the definition interpretation theorems (requiring existence), this allows an Isabelle syntax for introducing types that is equivalent to that given in Mizar. For brevity we present here only the conditional ones.

```
abbreviation (input) assume_mode_prefix
   ("assume _ mode _ → _ means _" [0,0,0,0] 10)
where "assume as mode M → type means cond ≡
   (M ≡ define_mode(λit. it be type & (as implies cond(it))))"

lemma assume_mode_property:
assumes df: "M ≡ define_mode(λx. x be D & (R implies P (x)))"
   and q: Q
   and assume_ex: "R ⟹ ex x being D st P(x)"
   and mode_ex: "ex x being D st True"
shows
   "(x be M iff (x be D & (R implies P(x)))) & Ex (λx. x be M)"
```

Attributes, which allow further restriction of types are treated equivalently: they are implications with more assumptions in the definitions in case of conditional definitions. The attribute definition details can be found in the formalization.

4 Background Knowledge

Mizar *background knowledge* are the facts that do not need to be explicitly given when justifying reasoning steps. Such facts can be associated to objects when the latter are defined. Relevant knowledge available in the article environment is automatically considered in the justification of any step whose statement incorporates objects associated with that knowledge. Mizar allows supplementing background knowledge using *properties* and *registrations*. Here we only introduce the syntax for the background knowledge, it is not implicitly used yet.

4.1 Property

A *property* is a specific characteristic which can be associated with a functor or with a predicate. For unary functors (functors having one visible argument)

the properties permitted by Mizar are projectivity ($\mathtt{f}(\mathtt{f}(\mathtt{x})) = \mathtt{f}(\mathtt{x})$) and involutiveness ($\mathtt{f}(\mathtt{f}(\mathtt{x})) = \mathtt{x}$). For a binary functor the properties known to Mizar are commutativity ($\mathtt{g}(\mathtt{x},\mathtt{y}) = \mathtt{g}(\mathtt{y},\mathtt{x})$) and idempotence ($\mathtt{g}(\mathtt{x},\mathtt{x}) = \mathtt{x}$). Finally, for predicates the properties are reflexivity, irreflexivity, symmetry, asymmetry, and connectedness (all binary). Clearly such properties can only be formulated, when the types of the arguments and result as stated in a particular property is valid.

Mizar allows writing such properties with the convenient syntax rather than unfolding such definitions, which additionally makes these properties known to the automation. For convenience we also provide input abbreviations that allow such syntax. Unfortunately apart from the name of the property and the object, we need to explicitly give the type of arguments, which Isabelle cannot infer on the Isar level. The abbreviations are analogous to those presented in the paper already, we show here only a few uses of such notations for brevity. Please consult the formalization for examples of all Mizar properties.

```
theorem xboole_0_def_8_asymmetry:
  "asymmetry set xboole_0_def_8"

theorem relat_1_def_7_involutiveness:
    "involutiveness Relation relat_1_def_7"
```

4.2 Registrations that Facilitate Type Inference

Mizar *registrations* are all features related to the automatic processing of type information. There are three kinds of registrations: clusters which aid the computation of types in the presence of adjectives, reductions which are special kinds of rewrite rules, and identify, which allows resolving conflicts that arise when Mizar attempts to identify objects based on their arguments.

There are three kinds of clusters:

- *existential registrations* used to retain type non-emptiness information. For example `funct_1_cl_2` states that there exists a set which is also a relation.
- *conditional registrations* used to extend the list of adjectives assigned to a term, given that particular adjectives have already been established. For example `relat_1_cl_0` states that every empty set is a relation. In particular, if the adjectives list is empty, a conditional registration allows establishing adjectives about any terms of given types. For example `relset_1_cl_1` establishes that every subset of a Cartesian product is a relation.
- *functorial registrations* used to automatically provide type information for compound terms. For example the singleton pair $\{[a, b]\}$ is both a relation and a function because of the clusters `relat_1_cl_7` and `funct_1_cl_3`.

We introduce the following notations, which allow introducing all three kinds of Mizar clusters in the Isabelle emulations with the same convenient syntax. Each actual cluster becomes an Isabelle lemma, which needs to be explicitly used in type checking.

```
abbreviation (input) cluster_prefix_existential
   ("let _ cluster _ for _" [10,10,10] 10)
where "let lt cluster attrs for type
      ≡ (lt ⟹ (Ex (λ X. X be attrs ‖ type)))"
```

```
abbreviation (input) cluster_prefix_conditional
   ("let _ cluster _ → _ for _" [10,10,10,10] 10)
where "let lt cluster attrs → attrs2 for type
      ≡ (lt ⟹ ( ⋀X. (X be type ∧ X is attrs) ⟹ X is attrs2))"

abbreviation (input) cluster_prefix_functorial
   ("let _ cluster _ → _" [10,10,10] 10)
where "let lt cluster fun → attrs
      ≡ (lt ⟹ fun is attrs)"
```

Reductions are another mechanism supported by Mizar in registration blocks used to enhance the capabilities offered by the properties shown in functor definitional blocks. Reductions are similar to rewrite rules that replace a term by its subterm, however Mizar does not perform the replacement, but rather creates an equality between the two terms. This equality can, but does not need to be used as part of the justification. Reductions allow arbitrary terms as left-hand sides, and are therefore more general than properties which are limited to Mizar patterns (functor with variable list). Mizer reductions are further discussed in [8].

Similarly to our emulation of Mizar properties, we rely on the user to state reductions as theorems. We again introduce an abbreviation that allows the convenient Mizar syntax of reductions and show an example of a reduction from the MML:

```
abbreviation (input) reduce_prefix
   ("let _ reduce _ to _" [10,10,10] 10)
where "let lt reduce term to subterm
      ≡ (lt implies (term = subterm))"

theorem relat_1_id_dom:
    "let X be set reduce dom (id X) to X"
```

5 Hidden Arguments

As already hinted in Sec. 3, a defined concept does not need to be explicitly applied to all the arguments declared in the definition block. This is possible, when Mizar can infer these arguments based on the types of arguments explicitly given in the pattern. Indeed, given the unique types associated with every term, it is possible to unambiguously infer the list of arguments occurring in this type.

Consider the function composition operation. Its result is clearly a function. We can justify, that the composition `p∘f` is of the type `X↦Z` given `f:X↦Y` and `p:Y↦Z`. Using a conditional definition, we can reduce the requirements concerning the type of `p`: namely that its range is a subset of `Z` and that the image of `f` is a subset of the domain of `p`.

The original Mizar definition of typed function composition is presented on the left and the Isabelle counterpart on the right. The Mizar formalization allows the user to omit `X`, `Y`, and `Z` in the term `p/*f`, as all three can be uniquely determined by types `Function` of `X,Y`, `Z-valued Function`. In a regular Isabelle definition it is not possible to omit any of the three variables, because the type system offered by the Isabelle framework on the Isar level is too weak to allow the inference of such hidden arguments.

```
definition
   let X,Z be set, Y be non empty set;
   let f be Function of X,Y;
   let p be Z-valued Function;
   assume rng f c= dom p;
   func p/*f -> Function of X,Z equals
      p*f;
end;
```

```
definition funct_2_def_11
  (" _ '/*'[_, _] _" [10,0,0,10] 90)
where
  "assume rng f c= dom p
   func p /*'[X, Z] f →
            Function-of X,Z equals
      p*'f"
```

We have so far avoided introducing ML-level procedures for definitions, which might allow us to reimplement some of Mizar's inference mechanisms. Without this, all variables that appear in the definition body, as well as the result in case of functors, and the mother type in case of modes, must be present in the pattern of the defined object. Thanks to the use of a let construction in the definitional theorem we can hide the variable Y. The variables X, Z must remain visible in the proposed approach.

Even if with the proposed approach some of the Mizar hidden variables remain visible in Isabelle, this does come in particularly useful in Mizar redefinitions. Almost all variables introduced in MML redefinitions are hidden arguments, and the proposed approach will allow keeping them all hidden. The definitions of Mizar objects in MML are often stated with little requirements given to the arguments that are necessary to create an object prototype, as well as a pattern for it. As more is proved about the object, it can be redefined making its type more precise.

Such redefinitions can be used to modify the actual definition, the type of a functor, or the mother type for a mode. Clearly, this is only possible if the modified definition or type can be proved from the more specific arguments. Therefore Mizar requires the user to prove certain correctness conditions appropriate for the kind of redefinition, in order to ensure that the definition is compatible with the original one. For a type redefinition the coherence condition is required, which ensures that the new object has the correct new type. For the modification of the definition body, the compatibility correctness condition is required, which states that the original one and the new one are equivalent under the new assumptions.

To enable redefinitions with the same correctness conditions as those required by Mizar, for each object kind we prove corresponding theorems. We show here only the ones for redefinitions of functors and modes, for the remaining ones refer to the file `Mizar_Defs.thy` in the formalization.

```
lemma redefine_func_means_property:
assumes lt: "lt"
assumes coherence: "F be M"
assumes compatibility: "⋀ it. it be M ⟹
            ((it = F) ⟷ newCondition(it))"
shows "F be M ∧ newCondition(F)"
  using coherence compatibility lt by auto
```

Note that proving the correctness conditions for a mode requires using the *Set axiom*, namely that objects are sets and that object is the root of the Mizar

type hierarchy. A redefinition becomes a regular theorem in Isabelle/Mizar, for example the redefinition of the range `rng` if the input argument is a `Function`:

```
theorem funct_1_def_3:
  "let f be Function
   redefine func rng f → set means
     (λ it. for y being object holds y in it iff
        (ex x being object st x in dom f & y = f . x))"
```

5.1 Hidden Arguments in Redefinitions

Providing redefinitions as theorems also helps in convenient support for new arguments in the redefinitions. Consider the definition of function application (MML article `FUNCT_1`) which given an arbitrary function `f` and object `x` returns the second element of the pair `[x,it]` contained in `f` if `x` belongs to the domain of the function, or the empty set otherwise.

```
definition
   let f be Function; let x be object;
   func f.x -> set means
   [x,it] in f if x in dom f otherwise it = {};
end;
```

The following redefinition performed in Mizar gives a more specific type of function application. Given that `f` is a function from the non-empty set `C` to the set `D`, and that the argument `c` is an element of `C`, the type of the functor can be made more precise, namely it returns an element of `D`. Note the additional arguments with respect to the original definition.

```
definition
   let C be non empty set, D be set;
   let f be Function of C,D;
   let c be Element of C;
   redefine func f.c -> Element of D;
end;
```

We could interpret a redefinition with additional arguments as a definition of a functor by equals, where the pattern contains all the variables present in the block (`C`,`D`,`f`,`c` in the example).

```
definition funct_2_def_5_2 (" _ , _ : _ . _" 190) where
  "func C, D : f . c → (Element-of D) equals f . c"
```

However, the argument `C` is not necessary neither in the functor type `Element of D` nor in the definition body `f.c`. In Mizar, the variable `C` is only used to specify the argument types of `f`. Using the Isabelle counterpart of the Mizar let construction (as we did in the initial experiments [6]) required the use of C in the pattern. By avoiding the let construction and providing the assumptions only in the interpretation of the definitional theorem, it is possible to reduce the arguments only to those present in the function body and result type (for functors) and mother type (for modes). Furthermore, the fact that a redefinition is a regular lemma allows us to finally hide also the variable `D`.

```
definition funct_2_def_5_1 (" _ : _ . _" 190) where
   "func D : f . c → (Element-of D) equals f . c"

theorem funct_2_def_5:
  "let C be non empty ||set & D be set &
       f be (Function-of C,D) & c be Element-of C
   redefine func f . c → (Element-of D)"
```

5.2 Hidden Arguments in Proofs

An important difference between the Mizar proofs and Isabelle's Isar proofs, is the fact that Isar requires the assumptions and the thesis for any proof block to be fixed in advance, whereas in Mizar various mechanisms (such as automatic unfolding of definitions) allows these to remain open. This means that when translating a reasoning from Mizar to Isabelle/Isar, even for specifying the block assumptions and thesis it is necessary to reconstruct the complete lists of hidden arguments, which would clearly hinder legibility. Most of the cases where this becomes an issue, are the proof blocks in which Mizar automatically unfolds definitions. The following example illustrates the necessary added arguments:

```
theorem funct_2_th_50:
  "for y be object, X be non empty ||set holds
     for f1,f2 be Function-of X,{y} holds f1=f2"
proof(intro ballI)
  fix y X f1 f2
  assume T0: "y be object" "X be non empty||set"
             "f1 be  Function-of X,{y}" "f2 be  Function-of X,{y}"
  show "f1 = f2"
  proof (rule  iffD2[OF funct_2_def_7[of X "{y}" f1 f2]])
```

In order to unfold the definition of equality in Isabelle, it is necessary to not only use the redefinition lemma `funct_2_def_7`, but also the complete list of the arguments which appeared in the redefinition block `X "{y}" f1 f2`.

```
theorem funct_2_def_7:
  "let A be set & B be set &
      f1 be Function-of A,B & f2 be Function-of A,B
   redefine pred f1 = f2 means
      for a be Element-of A holds f1 . a = f2 . a"
```

By reconstructing the arguments and adding them only in the cases where they are necessary, we can preserve readability while allow Isabelle to verify the proof scripts.

6 Conclusion

We have presented a number of components that make it convenient to use the Mizar object logic developed in Isabelle in a style that is similar to that offered by Mizar.

This includes mechanisms that allow all kinds of definitions available in Mizar, including conditional ones and making sure that the proof obligation required by Isabelle for each kind of definition corresponds to the Mizar's minimal requirements that guarantee the soundness of the defined *object*. We provided the relevant interpretation of the background information available in Mizar that is introduced using registrations and properties. We suggested mechanisms for Mizar's redefinitions, and provided a partial solution for hiding in Isabelle the arguments that are hidden in Mizar. Our work does directly generalize to other logical frameworks. However, as we do focus on Mizar-specific mechanisms, it is not clear whether it can be useful for logics not based on set theory with soft type system.

We tested the provided mechanisms in the process of further development of the Isabelle/Mizar library. We attempted to manually reformalize a number of subsequent articles in the MML. The current state of the Isabelle/Mizar formalization includes 65 Mizar-style definitions including 3 conditional ones, where 28 of them required Mizar-style justifications. In addition, we adopted all 10 properties originally formulated for these definitions in the MML. We further proved also 30 theorems, 26 registrations, 4 schemes, and 2 reductions. The total size of the development is 132kB, and it is available at:

`http://cl-informatik.uibk.ac.at/cek/cicm2017/`

6.1 Future Work

Mizar is abundant in mechanisms that allow for convenient input and manipulation of mathematical expressions. We have so far not considered imitating the notations used in the Mizar proof style, including the Mizar `let` and `take`. The abbreviations introduced focused on the first part of the MML, and we have not introduced any mechanisms for the two features not covered by the foundations and appearing only in later part of the library: Mizar structures and Fraenkel operators. A number of attributes and abbreviations introduced in the paper allow for much nicer manual input, but are not connected with any automation procedure so far. Finally, we have so far focused on using the Isar level as much as possible. Adding new outer syntax on the Isabelle/ML level, for example for definitions, would allow more hidden arguments, and would allow not separating the definition from the defining lemmas.

Acknowledgements

We thank Chad Brown for the discussions on the various set-theoretic foundations. This work has been supported by the ERC grant no. 714034 *SMART* and OeAD Scientific & Technological Cooperation with Poland grant PL 03/2016.

References

1. J. Abrial, M. J. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in event-b. *STTT*, 12(6):447–466, 2010.

2. J. Alama, M. Kohlhase, L. Mamane, A. Naumowicz, P. Rudnicki, and J. Urban. Licensing the Mizar Mathematical Library. In J. H. Davenport, W. M. Farmer, J. Urban, and F. Rabe, editors, *Intelligent Computer Mathematics (CICM 2011)*, volume 6824 of *LNCS*, pages 149–163. Springer, 2011.
3. G. Bancerek. On the structure of Mizar types. In H. Geuvers and F. Kamareddine, editors, *ENTCS*, volume 85, pages 69–85. Elsevier, 2003.
4. G. Bancerek, C. Byliński, A. Grabowski, A. Korniłowicz, R. Matuszewski, A. Naumowicz, K. Pąk, and J. Urban. Mizar: State-of-the-art and beyond. In M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge, editors, *Intelligent Computer Mathematics (CICM 2015)*, volume 9150 of *LNCS*, pages 261–279. Springer, 2015.
5. A. Grabowski, A. Korniłowicz, and A. Naumowicz. Four decades of Mizar. *Journal of Automated Reasoning*, 55(3):191–198, October 2015.
6. C. Kaliszyk, K. Pąk, and J. Urban. Towards a Mizar environment for Isabelle: Foundations and language. In J. Avigad and A. Chlipala, editors, *Conference on Certified Programs and Proofs (CPP 2016)*, pages 58–65. ACM, 2016.
7. C. Kaliszyk and J. Urban. MizAR 40 for Mizar 40. *J. Autom. Reasoning*, 55(3):245–256, 2015.
8. A. Korniłowicz. On rewriting rules in Mizar. *Journal of Automated Reasoning*, 50(2):203–210, February 2013.
9. A. Korniłowicz. Enhancement of Mizar texts with transitivity property of predicates. In M. Kohlhase, M. Johansson, B. R. Miller, L. de Moura, and F. W. Tompa, editors, *Intelligent Computer Mathematics – 9th International Conference, CICM 2016*, volume 9791 of *LNCS*, pages 157–162. Springer, 2016.
10. N. D. Megill. *Metamath: A Computer Language for Pure Mathematics*. Lulu Press, Morrisville, North Carolina, 2007. `http://us.metamath.org/downloads/metamath.pdf`.
11. S. Obua, J. D. Fleuriot, P. Scott, and D. Aspinall. ProofPeer: Collaborative theorem proving. *CoRR*, abs/1404.6186, 2014.
12. S. Obua, J. D. Fleuriot, P. Scott, and D. Aspinall. Type inference for ZFH. In M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge, editors, *Intelligent Computer Mathematics (CICM 2015)*, volume 9150 of *LNCS*, pages 87–101. Springer, 2015.
13. L. C. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science (1990)*, pages 361–386, 1990.
14. L. C. Paulson. Set theory for verification: I. From foundations to functions. *J. Autom. Reasoning*, 11(3):353–389, 1993.
15. K. Pąk. Improving legibility of formal proofs based on the close reference principle is NP-hard. *Journal of Automated Reasoning*, 55(3):295–306, October 2015.
16. F. Rabe. A logical framework combining model and proof theory. *Mathematical Structures in Computer Science*, 23(5):945–1001, 2013.
17. C. Schürmann. The Twelf proof assistant. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2009)*, volume 5674 of *LNCS*, pages 79–83. Springer, 2009.
18. J. Urban and G. Sutcliffe. ATP-based cross-verification of Mizar proofs: Method, systems, and first experiments. *Mathematics in Computer Science*, 2(2):231–251, 2008.
19. M. Wenzel, L. C. Paulson, and T. Nipkow. The Isabelle framework. In O. A. Mohamed, C. A. Muñoz, and S. Tahar, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2008)*, volume 5170 of *LNCS*, pages 33–38. Springer, 2008.
20. F. Wiedijk, editor. *The Seventeen Provers of the World*, volume 3600 of *LNCS*. Springer, 2006.