Isabelle Import Infrastructure for the Mizar Mathematical Library*

Cezary Kaliszyk[1][0000−0002−8273−6059] and Karol Pąk[2][0000−0002−7099−1669]

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

Abstract. We present an infrastructure that allows importing an initial part of the Mizar Mathematical Library into the Isabelle/Mizar object logic. For this, we first combine the syntactic information provided by the Mizar parser with the syntactic one originating from the Mizar verifier. The proof outlines are then imported by an Isabelle package, that translates particular Mizar directives to appropriate Isabelle meta-logic constructions. This includes processing of definitions, notations, typing information, and the actual theorem statements, so far without proofs. To show that the imported 100 articles give rise to a usable Isabelle environment, we use the environment to formalize proofs in the Isabelle/Mizar environment using the imported types and their properties.

1 Introduction

The Mizar project [10] has developed a language allowing users to write formal mathematics similar to the way it is done in informal mathematical practice [3]. This convenience for the users however means that the Mizar system is the only tool able to parse the language. Many exports of the *Mizar Mathematical Library* (MML) [1] to various formats and systems have been developed, including variants of XML [24,21], TPTP [6], and OMDoc [12], however they are all static. They allow inspecting the information contained in the Mizar library, presenting it [26], searching, and even give proof hints [20], but do not allow any further development of the proof scripts.

One of the reasons for this state-of-art is the architecture of the Mizar verifier. The verification of a Mizar proof script, called an *article* is performed by a series of independent programs, that each check particular aspects of syntactic and semantic correctness. Modules include derived information in the intermediate representation and drop the data which the Checker will not need to certify the proof. This final format, which is used by most of the exports, no longer contains the original user input and it is usually not possible to completely restore it.

Such lost information together with the processed script is necessary in our project [16] aiming to certify the Mizar proofs in Isabelle and create an environment to further develop the Mizar Library in the Isabelle logical framework [29].

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

For this, we develop an application able to combine the syntactic and semantic information available at the various stages of the Mizar processing. Such processed information is suitable for importing it in Isabelle or other frameworks, it could for example serve as a basis for a more complete import into OMDoc, that would include user steps and notations.

We develop an Isabelle infrastructure able to import the processed Mizar information into the Isabelle/Mizar object logic. Definitions of Mizar functions, types, predicates, etc. are automatically processed giving rise to Isabelle/Mizar constants together with their respective properties. Type inference information, contained in Mizar registrations and clusters is transformed to Isabelle theorems, that are furthermore put into special theorem lists that can be used by Isabelle/Mizar type inference. Actual theorem (as well as lemma and theorem scheme) statements give rise to corresponding statements in Isabelle with Mizar notations transformed into similar Isabelle MixFix notations [28].

Contributions This paper introduces an infrastructure for importing an initial part of the Mizar Mathematical Library into the Isabelle/Mizar object logic. The particular contributions are:

- We create a combined syntactic-semantic export of the Mizar proof data. As part of this we propose ways to match the processed semantic information with the original proof script.
- We develop an Isabelle infrastructure able to process and import the first 100 MML articles (out of 1318) into Isabelle/Mizar, so far mostly without proofs. The transferred values are loaded into the LCF environment in the spirit of Isabelle/Import [14].
- The imported environment allows users to work with Mizar data inside Isabelle, with features including notations and type inference. We demonstrate this by formalizing proofs depending on the imported parts of the library, namely the beginning of the Mizar article `NEWTON` which defines factorials and exponents and shows their basic properties.

Contents In Section 2 we discuss existing Mizar exports. Section 3 introduces the Isabelle/Mizar object logic. In Section 4 we discuss the Isabelle import infrastructure. Section 6 discusses combining the syntactic information. As this is mostly technical and not necessary to understand the previous sections, we chose to present it at this point. Finally in Section 5 we show an example formalization that uses the imported definitions, theorems, and Mizar type inference rules.

2 Existing Mizar Exports

This section introduces the process how Mizar processes a formalization, and discusses the existing exports from Mizar which use various parts of this process.

The Mizar system processes each formalization in multiple stages. The stages are independent processes, which communicate using external files. There are

more than 20 types of such files. This large number allows different stages and data exports to import only minimal information, which was crucial given the memory limits 40 years ago, when Mizar was conceived. In 2005, Urban [24] has modified the representation used in the intermediate stages to a slightly more accessible XML format. This has been done relatively late, as the larger representation does slow down the verification of the MML 1.67 times, but at the same time it allowed external access of some of the Mizar information.

tarski.xml

```
<Proposition line="27" col="35">
 <For pid="0" vid="2">
  <Typ kind="M" nr="1" pid="1"><Cluster/><Cluster/></Typ>
  <Is>
   <Var nr="1"/>
   <Typ kind="M" nr="2" pid="2"><Cluster/><Cluster/></Typ>
  </Is>
 </For>
</Proposition>
```

tarski.idx

```
<Symbol kind="I" nr="2" name="x"/>
```

tarski.eno

```
<Pattern kind="M" nr="1" aid="HIDDEN" formatnr="2" constrkind="M"
   constrnr="1" relnr="1">
```

tarski.frm

```
<Format kind="M" nr="2" symbolnr="2" argnr="0"/>
```

tarski.dcx

```
<Symbol kind="M" nr="2" name="object"/>
```

Fig. 1. The statement of the theorem "Everything is a set" in Mizar XML together with the information encoded in four other `tarski` XML files. These are necessary to decode the information. In the statement, the <For pid="0"vid="2"> binds the second occurrence of the symbol `x`, assigning it the type `object`, which corresponds to pid="1". Similarly the <Is> node encodes the information that the variable bound by the first `x` quantifier is of the type set pid="2".

The information processed by the first stages of Mizar is encoded in the Mizar XML (an example statement presented in Fig. 1), with some of the original syntactic information lost. All formulas are transformed to the Mizar normal form (MNF), which uses only selected logical connectives ($\wedge$, $\neg$, $\top$, and $\forall$) [4]. Furthermore, Mizar XML also uniquely identifies the constructors corresponding to particular notations, fixes their arities, and expands some abbreviations. For example the Mizar type `Function` of `A,B` is exported as `quasi_total Function-like Relation-like Element` of `bool [:A,B:]` in the XML representation.

Later versions of Mizar XML included `pid` attributes containing information about the notation and normalization, which allowed the generation of HTML

and a system for semantic browsing of the Mizar Library [4]. This information is passed by the Analyzer to the Checker, however the latter ignores it.

The notations are re-constructed based on this information heuristically, however in some cases it is not possible to guess the original representation.[3] Such small inconsistencies do not have a significant influence on the rendered HTML.

Urban has also developed the *Mizar Problems for Theorem Proving* (MPTP) export [25]. It is also based on the semantic XML representation. It aims to provide problems for automated reasoning, cross-verification [27] of Mizar, and more recently exploited for machine learning for theorem proving [13], and lemma extraction [19]. The export uses an XSLT stylesheet to make all problems independent from the Mizar environment, which allows consistent naming of symbols across the whole corpus. The core of Mizar is based on first-order logic, however the theorem and axiom schemes can also be exported to higher-order TPTP problems based on the same information [6].

The MPTP translation closely represents the first-order top-level statements of each article. Mizar types are represented as predicates. Definition blocks are unfolded to separate axioms stating the return type, the parent type, and the property of the introduced concepts. Similarly Mizar registrations (user-defined type inference rules) become axioms about the types. Second-order objects cannot be directly represented in the first-order format, therefore a fresh constant, together with an axiomatization, is introduced for each instance. This is theoretically justified by the set theoretic axioms of replacement and comprehension. This has been sufficient to translate the MML.

The MPTP representation was also used to import Mizar into OMDoc [12]. This means that the representation is semantically correct, but cannot reliably represent the original syntax. In particular many notations and variable names have been lost. Still this is enough to browse and search the Mizar library together with other theorem proving libraries.

Rather than extracting the information from the XML representation processed by the Checker, it is also possible to use the intermediate output of the parser. This approach has been used by *Weakly Strict Mizar* (WSX) [21] and *More Strict Mizar* (MSX) [7] which export subsequent syntactic layers. The aim of WSX (presented in Fig. 2) is to provide a standalone parser for Mizar articles, that can be used by external tools. MSX is an extension of WSX, where the representation is additionally augmented by variable information: variables are categorized as reserved, free, bound, or local constant together with unique identifiers and implicit quantifiers are made explicit. Both syntactic exports have complete information about the user input, but do not include any derived information. In particular the disambiguation and hidden arguments are missing, and are very hard to reconstruct for any external tool.

[3] These rarely affect the HTMLization of the current Mizar library, see for example http://mizar.uwb.edu.pl/version/current/html/funct_2.html#FC4 where the expression `K3(g,f)` includes `K3` rather than `f * g`. Note the reverse order or arguments.

```
<Proposition>
 <Label idnr="0" spelling="" line="27" col="5"/>
 <Universal-Quantifier-Formula line="27" col="5">
  <Explicitly-Qualified-Segment line="27" col="5">
   <Variables>
    <Variable idnr="2" spelling="x" line="27" col="7"/>
   </Variables>
   <Standard-Type nr="2" spelling="object" line="27" col="20"/>
  </Explicitly-Qualified-Segment>
  <Qualifying-Formula line="27" col="35">
   <Simple-Term idnr="2" spelling="x" line="27" col="28"/>
   <Standard-Type nr="1" spelling="set" line="27" col="35"/>
  </Qualifying-Formula>
 </Universal-Quantifier-Formula>
</Proposition>
```

Fig. 2. The statement of the theorem "Everything is a set" in Weakly Strict Mizar. WSX includes all user input, exactly as it was typed by the user and parsed in a tree form. The proposition without a label states that the explicitly universally bound variable `x` is of type `set`.

3 Isabelle and the Mizar Object Logic

Isabelle [29] is a logical framework, that is a proof assistant designed for the purpose of defining logics and working in these logics. The foundations of Isabelle are a variant of simple type theory with ML-style polymorphism. Isabelle's implementation is based on a relatively small kernel implemented in Standard ML. It includes functionality that makes it convenient to define constants present in logics, basic inference rules, notations, and the procedures specific to particular logics.

In our previous work we formally defined the semantics of Mizar as an Isabelle object logic [17]. This included mechanisms that allowed for the definition and use of Mizar types, introduction of meta-level constants and predicates, and the use of Mizar quantifiers. The notations of the resulting Isabelle/Mizar object logic have been optimized [15] allowing a combination of Mizar definitions with Isabelle-style ones, some hidden arguments, re-definitions, as well as extended constructions present in the Mizar library such as set comprehensions and structures. We finally defined a type inference mechanism for Isabelle/Mizar [18]. This Isabelle/Mizar object logic, serves as the basis for the imported Mizar library.

4 Isabelle Import Infrastructure

In this section we discuss the import of an already pre-processed Mizar library article. An outline of the procedure preparing an article for import will be discussed in Section 6.

Isabelle already includes a package that allows importing proofs originating from HOL Light and HOL4 [22,14] as well as one for importing the OpenTheory library articles [11]. Both of these are restricted to higher-order logic and only include higher-order logic kernel specific basic inference steps. The Mizar articles do not include any basic inference steps, but rather specify the different kind of Mizar language items: Mizar style definitions, actual proved statements or user proved type inference rules. The existing package are therefore insufficient for our purposes, in fact none of the Isabelle HOL/Import rules could be re-used.

We therefore propose a new format to represent Mizar article to import, which will consist of (possibly nested) sequences of commands. In practice, we will represent the article in XML with different tags corresponding to the possible Mizar proof script items.

The architecture of the Mizar import will be influenced by the Isabelle community's experience with HOL/Import. The first version of HOL/Import [22] attempted to syntactically create Isabelle theory files corresponding to the imported articles. It was soon realized, that the input syntax of Isabelle changes significantly with successive versions and it was too much work to keep the output of HOL/Import up to date with it. Therefore, the newer version of HOL/Import does not attempt to generate an Isabelle theory file source, but rather processes the values directly, giving rise to an environment with the definitions and theorems pre-loaded, together with documentation that a user can use instead of a theory file, to further develop the formalizations. This approach came to be a viable one, and did not need significant updates since its creation. In our current work we imitate the latter approach.

A common part shared by the import of the various Mizar item types are Mizar propositions, terms, and types all of which will be imported as Isabelle terms. To simplify this process, the term representation exported from Mizar will include all the necessary information to represent the knowledge in a logical framework. All variables are guaranteed to be quantified and include their meta-types (i.e., a Mizar type variable or a term variable or a propositional variable), as well as the meta-types of their arguments in case of higher-order variables (scheme variables). Applications and abstractions directly correspond to the Isabelle application and abstraction (abstraction is present in Mizar in case of constructions such as quantifiers, set comprehensions, or the choice operator). The seven constants corresponding to the basic predicate logic and five constant corresponding to the foundations of Mizar are mapped explicitly to their Isabelle counterparts. Finally, we also map manually the five constants corresponding to the axiomatic Mizar article `HIDDEN`. All subsequent constants will be defined by the imported Mizar articles.

Mizar definitions need to be presented in a format that can be accepted by Isabelle. This means equations, where left-hand side consists of the newly introduced concept possibly applied to some arguments. To fit the various kinds of Mizar definitions (including definitions of types, functions, predicates, and structures, conditional definitions, definitions by means, synonyms, and antonyms) in this format, we create a local theory context with assumptions, in that context

define the constant, and apply definitional theorems. Such theorems give rise to definition correctness obligations (such as for example existence and uniqueness for functions defined by means). The proof obligations are currently assumed – this will be discussed in future work. Finally, suitable derived theorems are declared and exported to the global theory and attributes (such as its use in the type inference mechanism) are applied.

Actual theorems and lemmas are straightforward to process, as we do not import the proofs yet. The statement is fully transformed to an Isabelle corresponding statement. There are two special cases of theorems. Mizar type inference rules (referred to as clusters in Mizar) need to be further processed and added to specific type inference lists. Mizar schemes give rise to higher-order theorems, therefore Mizar variable declarations need to be transformed to Isabelle assumptions.

The general methods for translating definitions will be discussed in Section 6.3. Here we will only give an example of a definition that provides two types of information, the meaning of the defined object ordinal2_def_10 and the Mizar type inference rule ordinal2_def_ty.

```
definition
  let fi be Ordinal−Sequence;
  given A such that
    A is_limes_of fi;
  func lim fi → Ordinal means
    it is_limes_of fi;
end;
```

```
thm ordinal2_def_10
  fi is Ordinal-Sequence_ORDINAL2M1 ⟹
  ∃A : Ordinal_ORDINAL1M3. A is_limes_of fi
    ⟹ lim fi is_limes_of fi
thm ordinal2_def_10_ty
  fi is Ordinal-Sequence_ORDINAL2M1 ⟹
    lim fi is Ordinal_ORDINAL1M3
```

5 Mizar-style Proof Development in Isabelle

In this section we present a case study, which shows the usability of the imported part of the Mizar library in Isabelle. For this, we manually re-formalize selected parts (1 reduction, 3 clusters, 6 theorems, and the definition proof obligations) of the Mizar article NEWTON, which defines powers and factorials.

The article is not among the first 100 imported articles, and our export infrastructure is not yet able to interpret the Mizar environments in order to process it automatically. The NEWTON article depends (directly or indirectly) on 80 Mizar articles. All these are imported automatically, therefore the current formalization relies both on Mizar and on Isabelle when it comes to proof checking. Importing the proofs will allow reducing the trust base to a single system.

In the paper we only present the statements, all the theorems have proofs in the formalization, and so does the definition, as one needs to prove that the result of the function is of the declared type. For correspondence, Isabelle theorem and constant names include their absolute MML addresses [25], we however consider using more canonical Isabelle-like names [2]. We first define the binary power function, Mizar version presented on the left for comparison:

```
definition
  let x be Complex,
      n be natural Number;
  func x |^ n → number equals
    Product (n |→ x);
```

```
mdef newton_def_1 (_ ^ _ [90,0]91) where
  mlet x is Complex,
       n is natural|Number
  func x^n → number equals
    Π (n ↦ x)
```

where $n \mapsto x$ denotes a constant sequence of length n, equal x everywhere.

Next, we prove a number of properties of the defined power operator. The proofs make use of various concepts (including natural numbers, complex addition, finite products) from the imported articles. Furthermore, the declared user type inference rules imported from previous articles are automatically used by the Isabelle/Mizar type inference, which automatically handles a number of reasonings about the soft types.

mtheorem newton_th_4:
$z^0 = 1$

mtheorem newton_th_5:
$z^1 = z$

mtheorem newton_th_6:
$z^{s +_{\mathbb{N}} 1} = z^s *_{\mathbb{C}} z$

mtheorem newton_th_7:
$(x *_{\mathbb{C}} y)^s = x^s *_{\mathbb{C}} y^s$

mtheorem newton_th_8:
$x^{s +_{\mathbb{C}} t} = x^s *_{\mathbb{C}} x^t$

mtheorem newton_th_9:
$(x^s)^t = x^{(s *_{\mathbb{C}} t)}$

An example of the use of an imported statement is the induction on natural numbers. It is used to show the user-level typing rule, that the power of a natural number is also a natural number.

```
registration
  let x, n be natural Number;
  cluster x|^n → natural;
  coherence proof
A0: n is Nat by TARSKI:1;
    defpred P[Nat] means
      x|^$1 is natural;
A1: for a being Nat st P[a]
      holds P[a+1]
    proof
      let a be Nat; assume P[a];
      then reconsider b = x|^a as Nat;
      x|^(a+1) = b*x
        by Th6;
      hence thesis;
    end;
A2: P[0] by RVSUM_1:94;
    for a being Nat holds P[a]
      from NAT_1:sch 2(A2,A1);
    hence thesis by A0;
  end;
end;
```

```
mtheorem
  mlet x is natural|Number, n is natural|Number
  cluster x^n → natural
proof
  have A0:n is Nat using tarski_th_1 by simp
  let ?P =
    λit. x^it is natural
  have A1: for a be Nat st ?P(a)
    holds ?P(a +_ℕ 1)
  proof(rule balll,rule impl)
    fix a assume [ty]:a be Nat and ?P(a)
    hence [ty]:x^a is natural|Number by mauto
    have x^a +_ℕ 1 = x^a *_ℂ x
      using newton_th_6[of a] by mauto
    thus x^a +_ℕ 1 is natural by mauto
  qed mauto
  have A2:?P(0) using newton_th_4 by mauto
  have for a be Nat holds ?P(a)
    using nat_1_sch_2[of ?P] A2 A1 by simp
  thus x^n is natural using A0 by simp
qed
```

The Isabelle/Isar environment with Mizar proofs imported is already usable, albeit the level of automation is still weaker than that of Mizar. Some of the Mizar abbreviations have been introduced as definitions, and are therefore not automatically unfolded. By specifying our own notations and additional registrations, the environment becomes well usable, where even some of the complicated proofs are of similar length to those of Mizar. It is however currently slower, especially when it comes to selecting the background information: in the above proof at some reasoning steps 139 typing judgements are derived automatically and stored in an Isabelle theorem list. This is possibly due to the highly optimized Mizar implementation, which uses arrays for all indexing and contrary to Isabelle does not need to rely on bi-resolution.

6 Combining the Syntactic and Semantic Representations of Mizar

In order to re-verify the Mizar proofs, we would like to export the individual proof steps faithfully. The semantic Mizar XML introduced in Sec. 2 is insufficient for many cases. In particular the term abbreviations, reservations, and hidden quantifiers, which are already supported by Isabelle/Mizar, are not preserved in Mizar XML, and a modification of the Mizar Checker to preserve such information would be a tremendous task, as the information would need to be correctly processed by all parts of the large Mizar kernel. A simple example of two formulas with *exactly* the same representation in XML is: $\alpha \to (\beta \to \gamma)$ and $(\alpha \wedge \beta) \to \gamma$. Even with the Mizar XML hints, the two are identical. This is a problem in Isabelle, as there the proof skeleton must precisely correspond to the proof [30]. A further discussion of combining the proof steps is provided in [23].

6.1 Procedure Overview

The combination of the syntactic and semantic Mizar processing data consists of two parts. First, we will match all the items from these both representations filling the missing information, using a process similar to that of the Mizar Analyser. Then we will ensure that the combined information corresponds to our Isabelle meta-model of Mizar [18], matching the constants and lists of arguments. In the following we present the process in more detail.

First, we match single items (both top-level items and individual proof items). Then we perform a full identification of all objects present in the item. Constructors for all ambiguous terms are determined, and argument lists are expanded by the hidden arguments computed by the Mizar Analyser. To match this with the transformed syntactic form, we modify the latter imitating some of the Analyser processes, including logical formula normalization, simplification of selected constructions (such as predicate and attribute negations). This also needs to identify the types of variables bound by the quantifiers and matching atomic propositions, and further their term and subterm arguments recursively. Additionally Mizar local abbreviations (Mizar syntax: `set`), which have been

```
<proposition label="tarski_th_1">
 <app>
  <logic id="Ball" type="o" args="2" argsType="ty_abs"/>
  <const id="HIDDENNM1" type="ty" args="0" argsType="set"/>
  <abs id="x" type="set" args="0">
   <app>
    <reservedconst id="IsType" type="o" args="2" argsType="set_ty"/>
    <var id="x" type="set" args="0" argsType="set"/>
    <const id="HIDDENNM2" type="ty" args="0" argsType="set"/>
   </app>
  </abs>
 </app>
</proposition>
```

Fig. 3. The formulation of the Mizar theorem "Everything is a set" with the combined syntactic and semantic representation. All terms are expressed using applications and abstractions of a meta-logic allowing an easy correspondence to the meta-model of Mizar, in particular making it easy to express in a logical framework.

fully unfolded in the XML format, can be identified and folded, which is useful to improve the legibility of proof texts. All Mizar constructions can be presented as applications of meta-constants, which is unique and allows a uniform way of processing the information, as was needed in the previous two sections. An example of the combined information is presented in Fig. 3.

6.2 Background Information

The processing of a Mizar article requires a precisely defined environment. In order to allow external processing, selected background information needs to be transformed and exported. In Mizar these are of three kinds: clusters which aid the computation of types in the presence of attributes, `reductions` which denote rewrite rules, and `identify`, which resolves conflicts that arise when identifying objects based on their arguments. Of course all of these are semantically theorems, and could be exported as such (as is done for example by MPTP [25]). Our import would however include its own type inference mechanisms, so we can export exactly the information Mizar has. In particular, we include constants corresponding to each kind of background information (the Isabelle/Mizar constructions are given in [15]). A limited use of this information was shown in Section 5.

An example exported cluster and presented in Isabelle is:

```
registration
  let f be one-to-one
        Function;
  cluster f" → one-to-one;
```

thm funct_1_cl_10
f is one-to-one ⟹
f is Function$_{FUNCT_1M1}$ ⟹
(f $^{-1}$) is one-to-one

which states that the inverse of a one-to-one function is also one-to-one.

6.3 Definitions

We export each individual definition separately (rather than using Mizar definition blocks) including all the available information for each definition. The data that is again distributed between the different semantic processed files and needs matching with the syntactic part to reconstruct the missing information. For example a redefinition of a Mizar function, whose type has been expanded, and the modification only concerns the right-hand side of the definition, the folded type is only present in the correctness condition.

All kinds of defined objects (meta level functions, Mizar types and predicates) require different information, however they share multiple common parts: a unique binding allowing refering to the definition body (such as `newton_def_1`), a unique constant definition, a notation, and the types of all the arguments of the defined object including their dependent types (Mizar allows term arguments in types), as well as the assumptions for conditional definitions. To preserve the Mizar semantics and at the same time allow for uniformity, Mizar attribute definitions need to have their last assumption transformed to an argument. For example the definition of an empty set uses `X` as an argument rather than an assumption to reflect the meta semantics:

```
definition let X be set;
  attr X is empty means
    not ex x st x in X;
```

```
mdef xboole_0_def_1 (empty) where
  attr empty for set means
    (λX. ¬ (∃x : object. x in X))
```

The definition body can usually be expressed directly, however many Mizar constructs allow definitions per cases. For those, specific abstractions need to be introduced, for example the (somewhat non-standard) definition of an element of a set in Mizar is formulated as follows, while its export includes all the information necessary to transform the cases to multiple implications:

```
definition
  let X be set;
  mode Element of X → set means
    it in X if X is non empty
          otherwise it is empty;
```

```
mdef subset_1_def_1 (Element _) where
  mlet X is set
  mode Element X → set means
    (λit. it in_TARSKIR2 X if X is non empty
     otherwise λit. it is empty)
```

The final exported information from definitions are the properties of the defined concepts, which can be introduced in Mizar definition blocks. Similar to registrations, they can be expressed as theorems. Mizar does not include a formulation of the properties, and we recover this meaning along with the syntax in the exported environment. The reason for this in Mizar, is that the concept cannot be used in the definition block. Therefore we recover the folded form of each property:

```
definition
  let X, Y be set;
  func X \/ Y → set means
    for x holds
      x in it iff x in X or x in Y;
  existence...
  uniqueness...
  commutativity;
  idempotence;
```

thm XBOOLE_0_def_3
  X is set ⟹ Y is set ⟹
    x in X ∪ Y ⟷ x in X ∨ x in Y

thm XBOOLE_0K2_commutativity
  ∀X : set. ∀Y : set. X ∪ Y = Y ∪ X

thm XBOOLE_0K2_idempotence
  ∀Y : set. Y = Y ∪ Y

6.4 Redefinitions

Mizar redefinitions allow expanding and clarifying the information about already existing objects. It is possible to modify any kind of object (function, predicate, or Mizar type). There are four main categories of redefinitions, and we will need to adapt the export process for each of the below categories:

1. Adding a notation to an object. The name and the visible arguments need to be preserved, however the new notation can include more hidden arguments.
2. Changing the definition body of an object. This is particularly important, as Mizar proofs must correspond to definitions, therefore showing the equivalence of two definitions allows for different proof obligations when the definition is expanded.
3. Making the type more precise. The result types of functions can be refined, as well as the mother type can be refined for Mizar types. For example the exponential function could be defined with range type being the type of real numbers. Later, the user can show that the result is also non-negative and inform the type system about it.
4. Adding selected properties to defined objects.

Each redefinition introduces a notation, equivalent to the original constant, with the additional arguments removed. Each modification of the definition body is formulated as a theorem. These can later be processed in Isabelle by modifying the default introduction and elimination rules, just like it is done for regular definitions.

```
definition
  let X, Y be set;
  redefine pred X = Y means
    X c= Y & Y c= X;
```

abbreviation
  X =$_{XBOOLE\_0R4}$ Y ≡ X = Y

thm xboole_0_def_10
  X is set ⟹ Y is set ⟹
    X =$_{XBOOLE\_0R4}$ Y ⟷ X ⊆ Y ∧ Y ⊆ X

Proving that the result of a function has a more precise type is semantically close to a Mizar cluster (the two however require different syntax in Mizar). It can of course be specified as a theorem, but we want to preserve both the syntax and the semantics in our export, therefore we annotate the syntax and verify

that the semantics are correct already in the imported setting and that such rules can be used by the type inference mechanism:

```
definition                                  thm finseq_1_add_3
  let p be FinSequence;                       p is FinSequence_FINSEQ_1M1 ⟹
  redefine func                               dom_RELAT_1K1 p is
    dom p → Subset of NAT;                      Subset_SUBSET_1M2 NAT_NUMBERSK1
```

Redefinitions that include properties, can be exported in a similar way to properties specified for original definitions. They give rise to uniquely identified theorems about the exported constants and their property names:

```
                                            abbreviation
definition                                    X misses_SUBSET_1R1 Y ≡ X misses Y
  let X, Y be non empty set;
  redefine pred X misses Y;                 thm SUBSET_1R1_irreflexivity
  irreflexivity;                            ∀Y : non empty | set.
                                              ¬ Y misses_SUBSET_1R1 Y
```

The Mizar language is very rich when it comes to notations. Notations are used for example for predicate synonyms and antonyms, function synonyms, or type abbreviations. Adding notations is very cheap, since all components of Mizar reason modulo a congruence closure algorithm. Rather than clarify these in the export, we imitate the Mizar syntax and semantics, by introducing a new constant along with its syntax for each such introduced notation. The new constant is defined to be equal to the previous one, modulo arguments. For example, the domain of a relation is the projection of the relation on its first component.

```
notation                                    abbreviation
  let R be Relation;                          dom_RELAT_1K1 R ≡ proj1 R
  synonym dom R for proj1 R;
```

Just like with attributes, to correctly preserve the Mizar semantics we would need to separate the last argument. However, as can be seen in the following example, that last argument can be removed in case of notations. To give complete semantics to an introduced type constant `new` defined in terms of an existing one `old`, rather than the Mizar `x is new for x is old` we need to define `new == old`. A concrete example for the Mizar type of uncountable sets is:

```
notation
  let X be set;                             abbreviation
  antonym X is uncountable                    uncountable_CARD_3V6 ≡ non countable
        for X is countable;
```

7 Conclusion

We have proposed a combination of the syntactic and semantic Mizar information, creating an export that includes all the information needed to

Theorems	5994
Lemmas	478
Definitions	1851
User typing rules	1517
Derived introduction and elimination rules	540
Object typing rules	448
Schemes	270
Definition Uniqueness	226
...	
Total imported facts	11920

Table 1. Facts available in the Isabelle/Mizar environment, subdivided into major fact categories.

import Mizar into a logical framework. We imported the first 100 articles of the library into the Isabelle/Mizar object logic, which gives rise to 11,920 Isabelle imported facts of different kinds (see Table 1). The imported library is usable for further development. The first articles from the exported Mizar, the import infrastructure, and our re-formalization are available at:
http://cl-informatik.uibk.ac.at/cek/cicm2018.tgz

The import process assumes a model of Mizar based on first order logic, therefore by defining a similar model of Mizar in another logical framework the import could be directly reproduced there. We have also exported most of the proofs, but various transformations are necessary to import it in Isabelle. So far, the import only verifies that the statements are consistent and type correct, only importing the proofs would allow complete re-verification of the Mizar library. This still poses a number of challenges. The Isabelle proof nested blocks are weaker than the Mizar `now` construction [30]. The exported structures generate a large number of selectors which we cannot process automatically yet. Finally, even if Isabelle includes strong general purpose automation [5], it may not be as powerful as the Mizar BY tailored for the Mizar proofs.

The main future work that goes beyond importing and certifying the whole Mizar library in Isabelle is to further develop the Isabelle/Mizar environment in order to allow more convenient proof development. This includes more Mizar-like infrastructure for notations that would allow overloading, disambiguation, and hidden arguments. Furthermore, Mizar includes many tools that allow optimizing complete proof scripts. We would like to generalize such tools to the level of a logical framework. The presence of two libraries with dependencies can allow improved proof automation [9]. Lastly, we also want to generate and provide documentation for the articles imported in Isabelle, which will allow browsing multiple prover libraries with proofs within one system [8].

References

1. Alama, J., Kohlhase, M., Mamane, L., Naumowicz, A., Rudnicki, P., Urban, J.: Licensing the Mizar Mathematical Library. In: Davenport, J.H., Farmer, W.M., Urban, J., Rabe, F. (eds.) Proc. 10th International Conference on Intelligent Computer Mathematics (CICM 2011). LNCS, vol. 6824, pp. 149–163. Springer (2011)
2. Aspinall, D., Kaliszyk, C.: What's in a theorem name? (rough diamond). In: Blanchette, J., Merz, S. (eds.) 7th Conference on Interactive Theorem Proving (ITP 2016). LNCS, vol. 9807, pp. 459–465. Springer (2016)
3. Bancerek, G., Byliński, C., Grabowski, A., Korniłowicz, A., Matuszewski, R., Naumowicz, A., Pąk, K., Urban, J.: Mizar: State-of-the-art and Beyond. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) Intelligent Computer Mathematics - International Conference, CICM 2015. LNCS, vol. 9150, pp. 261–279. Springer (2015)
4. Bancerek, G., Urban, J.: Integrated semantic browsing of the Mizar Mathematical Library for authoring Mizar articles. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) Mathematical Knowledge Management (MKM 2004). LNCS, vol. 3119, pp. 44–57. Springer (2004), https://doi.org/10.1007/978-3-540-27818-4_4
5. Blanchette, J.C., Greenaway, D., Kaliszyk, C., Kühlwein, D., Urban, J.: A learning-based fact selector for Isabelle/HOL. J. Autom. Reasoning 57(3), 219–244 (2016), http://dx.doi.org/10.1007/s10817-016-9362-8
6. 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.) Intelligent Computer Mathematics (CICM 2016). LNCS, vol. 9791, pp. 99–114. Springer (2016)
7. Byliński, C., Alama, J.: New Developments in Parsing Mizar. In: Jeuring, J., Campbell, J.A., Carette, J., Reis, G.D., Sojka, P., Wenzel, M., Sorge, V. (eds.) Intelligent Computer Mathematics - 11th International Conference, AISC 2012, 19th Symposium, Calculemus 2012, 5th International Workshop, DML 2012, 11th International Conference, MKM 2012, Systems and Projects, Held as Part of CICM 2012. 7362, vol. 5170, pp. 427–431. Springer (2012)
8. Corbineau, P., Kaliszyk, C.: Cooperative repositories for formal proofs. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM 2007. LNCS, vol. 4573, pp. 221–234. Springer (2007)
9. Gauthier, T., Kaliszyk, C.: Sharing HOL4 and HOL Light proof knowledge. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) 20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2015). Lecture Notes in Computer Science, vol. 9450, pp. 372–386. Springer (2015)
10. Grabowski, A., Korniłowicz, A., Naumowicz, A.: Four decades of Mizar. Journal of Automated Reasoning 55(3), 191–198 (2015)
11. Hurd, J.: The OpenTheory standard theory library. In: Bobaru, M.G., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NASA Formal Methods. LNCS, vol. 6617, pp. 177–191. Springer (2011)
12. 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)
13. Irving, G., Szegedy, C., Alemi, A.A., Eén, N., Chollet, F., Urban, J.: DeepMath - deep sequence models for premise selection. In: Lee, D.D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.) NIPS. pp. 2235–2243 (2016)
14. Kaliszyk, C., Krauss, A.: Scalable LCF-style proof translation. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) Interactive Theorem Proving (ITP 2013). LNCS, vol. 7998, pp. 51–66. Springer (2013)

15. 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.) Intelligent Computer Mathematics - CICM 2017. LNCS, vol. 10383, pp. 193–207. Springer (2017)
16. Kaliszyk, C., Pąk, K.: Progress in the independent certification of Mizar Mathematical Library in Isabelle. In: Ganzha, M., Maciaszek, L.A., Paprzycki, M. (eds.) Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017. pp. 227–236 (2017)
17. Kaliszyk, C., Pąk, K., Urban, J.: Towards a Mizar environment for Isabelle: Foundations and language. In: Avigad, J., Chlipala, A. (eds.) Proc. 5th Conference on Certified Programs and Proofs (CPP 2016). pp. 58–65. ACM (2016)
18. Kaliszyk, C., Pąk, K.: Semantics of Mizar as an Isabelle object logic, accepted by Journal of Automated Reasoning, http://cl-informatik.uibk.ac.at/cek/submitted/ckkp-jar17.pdf
19. Kaliszyk, C., Urban, J.: Learning-assisted theorem proving with millions of lemmas. Journal of Symbolic Computation 69, 109–128 (2015)
20. Kaliszyk, C., Urban, J.: MizAR 40 for Mizar 40. J. Autom. Reasoning 55(3), 245–256 (2015)
21. Naumowicz, A., Piliszek, R.: Accessing the Mizar library with a weakly strict Mizar parser. In: Kohlhase, M., Johansson, M., Miller, B.R., de Moura, L., Tompa, F.W. (eds.) Intelligent Computer Mathematics, CICM 2016. LNCS, vol. 9791, pp. 77–82. Springer (2016), http://doi.org/10.1007/978-3-319-42547-4_6
22. Obua, S., Skalberg, S.: Importing HOL into Isabelle/HOL. In: Furbach, U., Shankar, N. (eds.) International Joint Conference on Automated Reasoning (IJCAR 2006). LNCS, vol. 4130, pp. 298–302. Springer (2006)
23. Pąk, K.: Combining the syntactic and semantic representations of Mizar proofs, submitted
24. Urban, J.: XML-izing Mizar: Making semantic processing and presentation of MML easy. In: Kohlhase, M. (ed.) Mathematical Knowledge Management (MKM 2005). LNCS, vol. 3863, pp. 346–360. Springer (2005)
25. Urban, J.: MPTP 0.2: Design, implementation, and initial experiments. J. Autom. Reasoning 37(1–2), 21–43 (2006)
26. Urban, J., Bancerek, G.: Presenting and explaining Mizar. Electr. Notes Theor. Comput. Sci. 174(2), 63–74 (2007)
27. Urban, J., Sutcliffe, G.: ATP-based cross-verification of Mizar proofs: Method, systems, and first experiments. Math. in Computer Science 2(2), 231–251 (2008)
28. Wenzel, M.: Isabelle as document-oriented proof assistant. In: Davenport, J.H., Farmer, W.M., Urban, J., Rabe, F. (eds.) Intelligent Computer Mathematics - 18th Symposium. LNCS, vol. 6824, pp. 244–259. Springer (2011)
29. 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, 21st International Conference, TPHOLs 2008. LNCS, vol. 5170, pp. 33–38. Springer (2008)
30. Wenzel, M., Wiedijk, F.: A comparison of Mizar and Isar. J. Autom. Reasoning 29(3-4), 389–411 (2002)