Towards a Mizar Environment for Isabelle: Foundations and Language

Cezary Kaliszyk
University of Innsbruck
cezary.kaliszyk@uibk.ac.at

Karol Pąk
University of Białystok
pakkarol@uwb.edu.pl

Josef Urban
Czech Technical University in Prague
josef.urban@gmail.com

Abstract

In this paper we explore the possibility of emulating the Mizar environment as close as possible inside the Isabelle logical framework. We introduce adaptations to the Isabelle/FOL object logic that correspond to the logic of Mizar, as well as Isar inner syntax notations that correspond to these of the Mizar language. We show how Isabelle types can be used to differentiate between the syntactic categories of the Mizar language, such as sets and Mizar types including modes and attributes, and show how they interact with the basic constructs of the Tarski-Grothendieck set theory. We discuss Mizar definitions and provide simple abbreviations that allow the introduction of Mizar predicates, functions, attributes and modes using the Isabelle/Pure language elements for introducing definitions and theorems. We finally consider the definite and indefinite description operators in Mizar and their use to introduce definitions by "means" and "equals". We demonstrate the usability of the environment on a sample Mizar-style formalization, with cluster inferences and "by" steps performed manually.

Keywords Mizar, Isabelle, object logic, Isar.

1. Introduction

The Mizar system has been developed rather independently of the mainstream tactical proof assistants for more than forty years. The development has been started in 1973 by Andrzej Trybulec, who has been working on his PhD in topology, while simultaneously being very interested in linguistics. This focus has for a long time distinguished Mizar from other systems. The main concern has been from the very beginning to support working mathematicians in producing formally verified texts that resemble informal mathematics as much as possible. Over the forty years, this has led to the development of several mathematician-oriented features:

- Jaśkowski-style natural deduction[1], trying to approximate the way informal proofs are written
- rich term and formula language based on a dedicated parser allowing a number of complex notational constructs
- rather standard mathematical foundations: first-order logic and set theory
- linguistically motivated dependent soft type system and type inference mechanisms, completely independent of "computer-science inventions"[2] such as (typed) lambda calculus
- LaTeX-like compilation (batch) mode reporting many errors at once rather than an interactive mode
- a lot of research on the notion of logical "obviousness" [3, 24], motivating the fast Mizar refutational "by" checker which uniformly discharges all low-level proof obligations, rather than requiring the users to master – or even program – sophisticated custom tactics.

Combined with many experiments in formalizing more and more complicated mathematical texts, the system has been used for early development of one of the largest formal mathematical libraries. By 2002, the system allowed a rather faithful formalization of many parts of mathematics, the largest one being the collaborative formalization of most of the graduate Compendium of Continuous Lattices [2].

Following the "discovery of Mizar" during the QED project [22] discussions and workshops, Harrison has pioneered work on bringing some of the Mizar features to tactical proof assistants. This in particular concerns the declarative proof style ported in his Mizar Mode for HOL [5]. A number of such Mizar-like environments have later appeared in tactical proof assistants. However with the exception of Harrison's implementation of the Mizar "by" prover, they are usually limited to emulating to various extent the Jaśkowski-style natural deduction and its Mizar implementation. The foundations, the formula/term languages and the LCF-style tactics have typically not been influenced by Mizar. This is true also for the Isar language [31] and proof style predominately used in Isabelle today.

1.1 Contributions

In this work we start to develop a rather faithful emulation of the Mizar logic in the Isabelle logical framework (Section 2). We initially focus on emulating in Isabelle the Mizar term and formula language, including the Mizar soft typing constructs (Sections 3,4). We discuss Mizar definitional mechanisms and provide abbreviations that allow faithful introduction of Mizar predicates, functions,

[1] Isabelle/Pure as logical framework and Isabelle/Isar as proof language are based on a generalized form of Gentzen-style natural deduction.

[2] Some mathematicians tend to think that systems for formal mathematics should be designed by and for working mathematicians, not by and for computer scientists. Andrzej Trybulec was certainly one of them, occasionally being quite critical of making complicated science from what should be simple and intuitive for non-experts.

CPP '16, January 18–19, 2016, Saint Petersburg, Florida, USA.

ACM [to be supplied].
http://dx.doi.org/10.1145/???

attributes and modes using the Pure definition syntax, including the resulting proof obligations corresponding to Mizar (Sections 5,7). This already allows us to state the exact Mizar logical foundations – the Tarski-Grothendieck set theory (Section 6), and to port two basic Mizar formalizations to the new environment (Section 8). This includes porting of dozens of Mizar definitions and theorems, even though we still rely on the rather imperfect correspondence between the Isar and Mizar natural deduction systems, and on the rather imperfect correspondence between the Isabelle and Mizar automation procedures. Such issues and related future work are discussed in Section 9.

1.2 Related Work

There have been a number of attempts to translate the Mizar logic to various other formalisms, or to emulate parts of it. The largest translation of Mizar has been done by Urban [29] to the TPTP untyped first-order language. It preserves the proof semantics, and therefore can be used by the MizAR proof advice system [10], however does not preserve any of the user commands, notations, and the notations which we plan to look at. A partial-but-efficient emulation of the Mizar typing mechanisms has been implemented in the MoMM system [28] based on Schulz's E prover [25].

The first Mizar style proof mode for an LCF theorem prover has been created by Harrison [5], who also ported the (untyped version of) the Mizar "by" prover. Kunčar [12] has worked on recovering the Mizar system in the type system of HOL Light. As in here, the ultimate motivation was to eventually emulate sufficiently many Mizar mechanisms into an LCF-style proof assistant, however many parts are still missing.

The statements of the theorems in the whole Mizar Mathematical Library (MML) have been exported to the MMT logical framework [7]. This allows the use of various MMT services for MML, such as searching the library or providing proof advice, however does not include an independent verification of the proofs or proof automation.

Isabelle already has an object logic Isabelle/ZF [19] based on set theory. Already the foundational axioms of ZF differ from those of Tarski-Grothendieck, and the type system introduced by Mizar is very different from any of the existing object logics in Isabelle. Furthermore, the library of Isabelle/ZF and the automation provided is quite different from our emulation.

Agerholm and Gordon [1] compare the possible representations of ZF set theory in HOL and Isabelle. The approach to implement set theory as an object logic over HOL has been further extended by Obua in HOLZF [16]. There, the axioms of set theory are added on top of higher-order logic. This has been further explored by his ProofPeer system, where the underlying higher-order logic is limited to a minimum [17].

Finally there are many automated translations between proof assistants, including the import of proofs from OpenTheory [6] and HOL Light including Flyspeck [9] to Isabelle/HOL, as well as an interpretation of Isabelle/HOL proofs in Isabelle/ZF [11].

2. Preliminaries

2.1 Logical Frameworks and Isabelle

Most proof assistants are based on a single fixed logic. This is in contrast to logical frameworks, which are designed to allow expressing various logics and provide practical support for reasoning in these logics. The most known logical frameworks are LF (with its most current implementations being Twelf [26] and MMT [23]) and Isabelle [33].

Isabelle is a generic theorem prover designed to support a variety of different logical systems. Its kernel, implemented in Standard ML, provides a meta-logic called "Pure", which is a version of simple type theory with shallow polymorphism. This meta-logic presents the user with a very limited number of logical operators necessary to implement object logics: equality, implication, conjunction, and the universal quantifier.

A number of object logics have been implemented in Isabelle, including Isabelle/HOL, Isabelle/ZF [19], Isabelle/TLA [14], Isabelle/CTT, and Isabelle/LCF. Paulson [18] provides an introduction to the various object logics implemented in Isabelle, discussing the choices made when implementing the object logics.

2.2 Mizar

The current incarnation of Mizar has developed since 1973 from many experiments with various parts of the system [13], including the foundations, type system, syntax and proof checking. In 1989, starting with the focus on the development of the MML, most of these parts have become rather fixed.

The logic is classical first-order logic with a minimal addition of second-order schemes (schematic axioms and theorems) that need to be explicitly instantiated by the user. *Types* are not foundational in this logic. Semantically, types are just first-order predicates with some automation attached. Such automation typically consists of user-programmable Horn clauses [15, 28], propagating various (parameterized) properties (adjectives) in the Mizar terms in a bottom-up way. As in other typed systems, Mizar types are also used for early type-checking, i.e., prohibiting expressions that do not type.

The proof system is Jaśkowski-style natural deduction [8, 21], complemented with a very limited, but fast (and type-aware) refutational first-order prover – the Mizar by [34]. The language also includes constructs for (conservatively) defining predicates, functions, and types, stating (and proving) their properties such as reflexivity and commutativity, and declaring (and proving) the type-automation clauses.

This logic could be used with various axioms, but in MML everything is based on the axioms of the Tarski-Grothendieck set theory. This is a proper strengthening of ZFC, adding the Tarski's axiom A [27] (the last axiom in Fig. 1). This axiom implies the Axiom of Choice (AC) and provides arbitrarily large strongly inaccessible cardinals and thus models of ZFC, which is used to avoid proper classes in some formalizations. In addition to AC, Mizar adds a global choice function. Related to the soft type system is an extensive notational system, allowing both ad-hoc and parametric overloading, synonyms, antonyms, complex symbols, and other linguistic features such as various fixities and argument hiding for closely mimicking mathematical vernacular.

3. Mizar Logic in Isabelle

In order to provide in Isabelle the basic Mizar language, we first import the Isabelle object logic FOL, and augment it with the Mizar logical notations implies, iff, or, and not. We also define an if-then-else construction used inside Mizar definitions (written in Mizar if-otherwise).

The Isabelle object logic FOL defines a type of Boolean propositions o. We additionally introduce two Isabelle types necessary for the Mizar foundations: an Isabelle type for Mizar concrete set instances s and an Isabelle type of Mizar type expressions ty. A value of the type ty in Mizar is formally either an *adjective cluster*[3] or a full *type expression*[4]. This means that it can also be a bare *attribute* (such as empty), a single *adjective* (such as non empty), or

[3] http://mizar.org/language/syntax.html#Adjective-Cluster

[4] http://mizar.org/language/syntax.html#Type-Expression

a bare *radix type*[5] (constructed from a parameterized *mode*[6] such as Element of NAT). The construction of Mizar-like compound types will be performed on the Isabelle term level inside the ty type. When the context is clear we will say just (Mizar) *type* instead of (Mizar) type expression, and *cluster* instead of adjective cluster.

Mizar types are semantically predicates over sets, however the syntactic constructs available for Mizar types differ a lot from those used for Mizar predicates. Mizar types can occur as arguments of certain constructions (is, be, being, and the), while Mizar predicates are directly applied to terms. We introduce the same distinction using the Isabelle type system. This distinction is introduced together with constants that let us convert between the two (attribute and typexp will convert predicates over sets to Mizar types, and prefix_is will verify that the given set satisfies the predicate represented by the type). In certain Isabelle object logics (most notably HOL, ZF, and CCL) a similar distinction between sets and predicates is introduced using two constants (Collect and member or mem). This has a number of advantages, the most important being stronger automation. (In the context of Isabelle/HOL this separation is also necessary for general code generation [4]).

typedecl s
typedecl ty

We next introduce the Isabelle meta-level constants. These constants will be axiomatized, with the following intentional meanings. tys allows the construction of Mizar types and clusters from adjectives (or from other clusters and radix types). Note that this is more liberal than in Mizar: only a single radix type is allowed there in each type expression.[7] This more liberal setting (free mixing of types and clusters) applies also to the following constants. non-ty constructs the negated adjective from an attribute. prefix-is given a set and a type returns a proposition, which is valid if the set belongs to the Mizar type (checking this is equivalent to the application of the predicate represented by the Mizar type). Attributes and modes are built using the constants attribute and typexp respectively. Their Isabelle types are the same, but modes require the existence (non-emptiness) precondition, which will we will discuss when introducing the axiomatization below. Finally, we introduce the definite and indefinite description operators the1 and prefix-the, with the former defined for any predicate, and the latter defined only for Mizar types.

consts
  tys :: ty ⇒ ty ⇒ ty
  non-ty :: ty ⇒ ty
  prefix-is :: s ⇒ ty ⇒ o
  attribute :: (s ⇒ o) ⇒ ty
  typexp :: (s ⇒ o) ⇒ ty
  the1 :: (s ⇒ o) ⇒ s
  prefix-the :: ty ⇒ s

We next introduce the axioms that define these constants. In the following axiomatization we will already use the infix syntax is for the constant prefix-is, the syntax the for the constant prefix-the and the empty infix constant _ for the constant tys. The way these notations are properly introduced without introducing a high-level of syntax ambiguity will be explained in Section 4. The meanings of the below axioms are as follows: A set is in a compound Mizar type if it satisfies all the adjectives and the radix type. A set belongs to the negated adjective non when it does not satisfy the underlying adjective. Adjectives and radix types correspond to predicates, but the latter require existence. Finally the description operators are axiomatized as usual.

axiomatization where
  tys[simp]: x is d1 d2 iff x is d1 & x is d2 **and**
  non-ty[simp]: x is non d iff not x is d **and**
  attr-spec[simp]:
    A = attribute(P) ⟹ (x is A) iff P(x) **and**
  typexp-property:
    A = typexp(P) ⟹
      ∃x. P(x) ⟹
        x is A iff P(x) **and**
  the1-property:
    ∃x. P (x) ⟹
      (∀x y. P (x) ∧ P (y) implies x = y) ⟹
        P (the1 (P)) **and**
  the-property[simp]: ∃x. x is d ⟹ (the d) is d

As the Mizar types are now introduced, we can also introduce the Mizar quantifiers. Mizar universal and existential quantification requires specifying the Mizar type of the introduced bound variable. For this we introduce two constants Ball and Bex. We define them in a very similar way to the bounded quantifiers present in Isabelle/HOL and Isabelle/ZF, however instead of predicates we use the given Mizar types. We additionally introduce the usual automation for these quantifiers (introduction and elimination rules with the Isabelle attributes [intro!], [dest], etc).

definition Ball :: ty ⇒ (s ⇒ o) ⇒ o **where**
  [simp]: Ball(D, P) iff (∀x. x is D implies P(x))
definition Bex :: ty ⇒ (s ⇒ o) ⇒ o **where**
  [simp]: Bex(D, P) iff (∃x. x is D & P(x))

4. Notations

In order to make the emulated environment more similar to the Mizar system, we further introduce notations for the basic constants axiomatized in the previous section that correspond to the Mizar notation.

We first need to introduce a syntax for Mizar types. The Isabelle system provides an extensible parser for the inner syntax, which is based on context free priority grammars [32]. It is possible not only to add terminals to the grammar, but also to add nonterminals together with production rules for them, as well as rewrite rules that work on the parse trees that are separate from the grammar (these rewrite rules in the form of parse and print translations are denoted by ⇀ and ↽). We will use these mechanisms to allow expressions such as x is non empty set in such a way that it does not clash with other notation. For this we introduce a syntax nonterminal for Mizar compound types tys. We will use this nonterminal in the syntax of constants that allow compound types and translate the occurrence of this nonterminal to the constant tys. We introduce both be and is as infix notations for prefix_is together with appropriate precedences.

nonterminal tys
syntax
  Mizar.prefix-is :: s ⇒ tys ⇒ o (**infix** be 90)
  Mizar.prefix-is :: s ⇒ tys ⇒ o (**infix** is 90)
  Mizar.prefix-the :: tys ⇒ s (the - [79] 80)
   :: ty ⇒ tys (-)

[5] http://mizar.org/language/syntax.html#Radix-Type

[6] The distinction between, e.g., a mode and radix type (or an attribute and an adjective) is the same as between a functor and a term or a predicate and an atomic formula. The former is always the *constructor* of the latter *logical object*. The terminology is however often mixed up.

[7] There have been many discussions in the Mizar team about merging the concepts of attributes and modes when the former started to allow explicit (visible) arguments in the same way as modes do. However having only a single "most specific radix type" is still a significant part of the Mizar type-checking algorithms.

```
  -tys :: ty ⇒ tys ⇒ tys (- - [90,90] 100)
translations
  -tys(d, ds) ⇌ CONST tys(d, ds)
```

Now the following are valid Isabelle/Mizar expressions, the constants tys and prefix-non are inserted in the appropriate places and the notations do not clash with the usual Isabelle function application.

```
term x is set
term x is empty set
term x is non empty Element of NAT
term x is one-to-one non onto Function of A,B
term the empty set
```

We also introduce three syntax nonterminals representing a group of variables, a typed group of variables and a group of typed variable groups, together with syntax translations that allow the usual Mizar quantifier format for-holds and ex-st. The type given for a typed variable group is duplicated for every variable in the group. We finally introduce input-only syntax for Mizar's reverse meta-implication provided used for formulating schemes.[8]

With these notations, the following are allowed by the Isabelle parser, and they have their intentional meanings: the quantifiers are interpreted as Ball and Bex with appropriate Mizar types. Thanks to the use of the Isabelle syntax translation mechanisms, these are also printed in the Mizar syntax in the Isabelle output (including printed goal-states).

```
term P & Q iff Z implies (not P implies R or W)
term for x,y being set,z being object holds P(x,y,z)
term ex y being even Element of NAT st Q(y)
term P provided Q
```

5. Between Logic and Set Theory

The first article in the Mizar Mathematical Library, called Hidden, is an axiomatic (not verified by the Mizar checker) article which introduces the basic concepts needed in an axiomatization of set theory: sets, set membership, and equality. The article first introduces two Mizar modes: object and set. set is a subtype of object. The type object is a relatively new addition to Mizar's types. It was introduced to hide some "objects" from the Mizar type automations concerning sets. For example, often one does not want to know that real numbers are sets with some internal structure. However, it is still provable in Mizar that everything is a set.

These modes are translated to an Isabelle representation, together with an implication between them and the existence of the elements in them. The first interesting difference between our formalization and Mizar, is the fact that Mizar axiomatizes the existence of an object and of a set, while we only axiomatize the former and will be able to prove the latter in the next Section, Sec. 6.

```
axiomatization
  object :: ty and
  set :: ty
where
  object-exists[simp]: ex x being object and
  hidden-mode[simp,intro]: x is set ⟹ x is object
```

Mizar next introduces the constant equality and axiomatizes it as a reflexive and symmetric relation. This is not enough to axiomatize equality in a logical framework, as without substitutivity we cannot even derive transitivity of equality. Furthermore, it would not permit the use of the Isabelle simplifier, and as we do not implement any Mizar-like automation, this would make the use of the emulated system very cumbersome. Therefore we re-use the equality of Isabelle/FOL, and only introduce the notation for inequality used by Mizar:

```
abbreviation
  not-equal :: s ⇒ s ⇒ o  (infix <> 50) where
    x <> y ≡ not (x = y)
```

Finally the article introduces the set membership constant in, whose properties will be defined by the Tarski-Grothendieck axiomatization discussed in the next section.

```
consts
  prefix-in :: s ⇒ s ⇒ o (infix in 50)
```

6. Tarski-Grothendieck Set Theory

We can now proceed with the introduction of the Tarski-Grothendieck axioms of set theory. The Mizar code of the articles TARSKI_0 (basic axioms of set theory) and TARSKI_A (Tarski's Axiom A) is presented alongside with our Isabelle axiomatization in Fig. 1. There are several differences between the two axiomatizations, necessitated by the first version of the setting which we use to emulate Mizar in Isabelle. They are as follows:

- Our Isabelle emulation so far needs to explicitly mention all the variables. This is visible for example in tarski-0-2, where the explicit quantification over X and Y is necessary. In Mizar these free variables become explicitly universally quantified at the outermost scope.
- Mizar uses the reserve mechanism to remember the default types of free variables. Mizar typed quantifications are now internally represented as Isabelle predicates (assumptions in case of universal quantifiers), and we do not yet have a mechanism similar to Mizar reservations in place. Therefore we explicitly use the being syntax in every quantifier. This is visible in tarski-0-5, where we need to say ex Y being set st in place of ex Y st.
- In order for the scope of quantifiers to maximally extend to the right, their Isabelle precedence is very low. This perfectly matches the Mizar desired bindings, but requires additional parentheses around quantifiers in more complex logical formulae.
- The application of a predicate to arguments uses the same syntax as the Isabelle usual application syntax, namely P(x, y) rather than the Mizar P[x, y].
- A Mizar scheme becomes a regular theorem (or axiom), but the type requirements of the scheme A() -> set become an Isabelle theorem assumption A is set. This can be seen on the TARSKI_0:sch 1 scheme.

With the first Mizar axiom, saying that every object is a set, we can now prove that there exists a set:

```
theorem set-exists[simp]: ex x being set
```

This is a straightforward consequence of object-exists, the-property, and tarski-0-1:

```
proof−
  have (the object) is set
    using object-exists the-property tarski-0-1 by auto
  thus ?thesis by auto
qed
```

[8] http://mizar.org/language/syntax.html#Scheme-Block

```
:: Set axiom
theorem :: TARSKI_0:1
    for x being object holds x is set;

:: Extensionality axiom
theorem :: TARSKI_0:2
    (for x being object holds x in X iff x in Y) implies X = Y;


:: Axiom of pair
theorem :: TARSKI_0:3
    for x,y being object ex z being set st
        for a being object holds
            a in z iff a = x or a = y;

:: Axiom of union
theorem :: TARSKI_0:4
    for X being set
        ex Z being set st
            for x being object holds
                x in Z iff ex Y being set st x in Y & Y in X;

:: Axiom of regularity
theorem :: TARSKI_0:5
    x in X implies ex Y st Y in X &
        not ex x st x in X & x in Y;


:: Fraenkel's scheme
scheme :: TARSKI_0:sch 1
Replacement { A() -> set, P[object,object] }:
    ex X st for x holds
        x in X iff ex y st y in A() & P[y,x]
provided
    for x,y,z being object
        st P[x,y] & P[x,z] holds y = z;

:: Tarski's axiom
theorem :: TARSKI_A:1
    ex M st N in M &
        (for X,Y holds X in M & Y c= X implies Y in M) &
        (for X st X in M ex Z st Z in M &
            for Y st Y c= X holds Y in Z) &
        (for X holds X c= M implies
            X,M are_equipotent or X in M);
```

```
— Set axiom
axiomatization where tarski-0-1:
  for x being object holds x is set

— Extensionality axiom
axiomatization where tarski-0-2:
  for X,Y being set holds
    (for x being object holds x in X iff x in Y) implies X = Y

— Axiom of pair
axiomatization where tarski-0-3:
  for x,y being object holds ex z being set st
    for a being object holds
      a in z iff a = x or a = y

— Axiom of union
axiomatization where tarski-0-4:
  for X being set holds
    ex Z being set st
      for x being object holds
        x in Z iff (ex Y being set st x in Y & Y in X)

— Axiom of regularity
axiomatization where tarski-0-5:
  for x being object,X being set st
    x in X holds (ex Y being set st Y in X &
      not (ex z being object st z in X & z in Y))

— Fraenkel's scheme
axiomatization where tarski-0-sch-1:
  A is set ⟹ (
    ex X being set st for x being object holds
      x in X iff (ex y being object st y in A & P(y, x))
  provided
    for x,y,z being object
      st P(x, y) & P(x, z) holds y = z)

— Tarski's axiom
axiomatization where tarski-a-th-1:
 for N being set holds ex M being set st N in M &
  (for X,Y being set holds X in M & Y c= X implies Y in M) &
  (for X being set st X in M ex Z being set st Z in M &
      (for Y being set st Y c= X holds Y in Z)) &
  (for X being set holds X c= M implies
      X,M are-equipotent or X in M)
```

Figure 1. TARSKI_0 and TARSKI_A in Mizar and Isabelle/Mizar

7. Mizar Function Definitions

Given the axiomatic Mizar articles which we translated in the previous two sections, all other articles in the Mizar Mathematical Library are fully checked by the Mizar checker. The first non-axiomatic article, TARSKI, introduces the most basic definitions of set theory, such as a singleton set, a set that contains precisely two elements, a union of sets, etc.

Mizar definitions are written in definition blocks[9], which may contain multiple declarations of the types of parameters (*loci* declaration), optional assumptions, and proper definitions using these declarations. Assumptions are used when the type system is not strong enough to capture all the definedness conditions just by the loci declarations.

The first kind of definition we are interested in Mizar is the functor definition[10] of meta-level functions (*functors* in the Mizar terminology), which use the syntax func. There are two kinds of meta-level function definitions: definitions by means and by equals. In both cases, the functor pattern (e.g. $\{x\}$), its result type and the definiens need to be specified. The general method is a definiens using means, providing a definitional formula for the defined functor. Apart from the bound variables introduced inside the definiens, the definiens can only use the parameters introduced in the current definitional block, together with a special variable it, which marks the occurrences of the functor pattern that is being defined. Mizar creates from the definiens a definitional theorem, where every occurrence of it is replaced by the defined functor pattern. Mizar also automatically creates the existence and uniqueness

[9] http://mizar.org/language/syntax.html#Definition-Block

[10] http://mizar.org/language/syntax.html#Functor-Definition

proof obligations which need to be proved by the author so that the definitional extension is conservative.

Often the functor definiens has the form it = ... , where it does not occur on the right-hand side. In this case a simplified method using the keyword equals can be used. The definiens is then just a term, and the only possible proof obligation is showing that the term has the correct result type.

In order to provide both kinds of definitions in our emulation along with a syntax that resembles that of Mizar, we provide the following two kinds of notation:

```
abbreviation (input) means-prefix
   (let - func - → - means - [0,0,0] 10)
where let lt func def → dom means cond ≡
  def = the1 (λit. lt implies (it is dom & cond(it)))

abbreviation (input) equals-prefix
   (let - func - → - equals - [0,0,0] 10)
where let lt func def → dom equals exp ≡
   def = the1 (λit. lt implies (it is dom & it = exp))
```

This allows introducing definitions in the usual Mizar notation let lt func def → dom means cond. It works as follows. Given certain constraints lt (type constraints or assumptions) it constructs an equality between the newly defined term def and the description operator applied to the implication between the constraints and the desired definitional property cond. Since the definition needs to introduce the variable used in the abstraction, namely it, the user will typically write the definitional property as a lambda abstraction λit. P(it). The introduction of it in the user input could be avoided with the help of an Isabelle/ML syntax translation, albeit making the notation itself much less readable.

In order to let users obtain the typing rules, the definitional properties and the uniqueness properties implied by user definitions introduced using func-means and using func-equals we provide the following two theorems:

```
lemma means-property:
assumes df: f = the1(λx. Q implies x is D & P(x))
 and q: Q
 and ex: ex x being D st P (x)
 and un: ⋀x y. x is D ⟹ y is D ⟹
     P (x) ⟹ P (y) ⟹ x = y
 shows f is D & P(f) & (x is D & P(x) implies x = f)

lemma equals-property:
assumes df: f = the1(λx. Q implies x is D & x=g)
 and q: Q
 and coherence: g is D
 shows f is D & f = g
```

The way these two theorems are currently used is by instantiating them with the definition and the assumptions of the definition. They then leave the same proof obligations as those required by Mizar.

In Fig 2 we present a few first selected Mizar definitions together with their Isabelle counterparts. Each definition is followed by a theorem that states all the information derived by our emulation from the user provided obligations. For example in the case of a definition by fun-means this means, that given the proofs of existence and uniqueness, the typing rules, the definitional property and the uniqueness for the definition are stated. In order to omit the necessity to state such properties for each constants manually, we use the schematic_theorem mechanism of Isabelle - the stated properties are automatically derived from the conclusion of the appropriate definitional theorem. For example in the case of the definition of a singleton set (tarski-def-1), given the user-provided proofs of existence and uniqueness:

```
show ex X being set st for x being object
  holds x in X iff x = y
proof−
  obtain X where
    A1: X is set & (for x being object holds
      (x in X iff (x = y or x = y)))
        using tarski-0-3 assms by blast
  then have X is set &
    (for x being object holds x in X iff x = y) by auto
  thus ?thesis by auto
qed

fix X1 X2
assume A1: X1 is set and A2: X2 is set and
      A3: for x being object holds x in X1 iff x = y and
      A4: for x being object holds x in X2 iff x = y
{
  fix x
  assume Z1: x is object
  have x in X1 iff x = y using Z1 A1 A3 by auto
  hence x in X1 iff x in X2 using Z1 A2 A4 by auto
}
thus X1 = X2 using tarski-th-2 A1 A2 by blast
```

The following three Isabelle lemmas become available:

```
y is object ⟹ {y} is set

y is object ⟹ x is object ⟹ x in {y} iff x = y

y is object ⟹
x is set ⟹ for xa being object holds xa in x iff xa = y
⟹ x = {y}
```

With such properties available for all defined constants it is possible to start proving the various characteristics of the constants specified in Mizar. For most theorems the Mizar statements are given in full, and we can prove precisely the same properties in our emulated Mizar system. However, for a few properties Mizar uses mathematical names to describe properties, such as commutativity or asymmetry. In such cases our formalization at the moment needs to manually state the such theorems, for example:

```
theorem tarski-def-2-commutativity[simp]:
  for x,y being object holds {x,y} = {y,x}

theorem prefix-in-asymmetry[simp]:
  for x,X being set holds not (x in X & X in x)
```

With this we can finish the translation of the Mizar article TARSKI.

8. Schemes, Attributes, and the Hilbert Operator

We next proceed with the Mizar article XBOOLE_0. The article starts with a proof of a scheme. From our emulation point of view, the scheme is a regular theorem, and we just use the Mizar keyword provided for stating its assumptions, enriched in our case by the type assumptions.

```
theorem xboole-0-sch-1:
  ex X being set st for x being object holds
     x in X iff x in A & P(x)
provided
   A is set
```

We also show the proof of this scheme to show the adaptations we needed to make to all the Mizar proofs:

```
definition
    let y be object;
    func { y } -> set means
:: TARSKI:def 1
    for x being object holds x in it iff x = y;

definition
    let X;
    func union X -> set means
:: TARSKI:def 4
    x in it iff ex Y st x in Y & Y in X;

definition
    let x,y be object;
    func [x,y] -> object equals
:: TARSKI:def 5
    { { x,y }, { x } };
```

```
definition tarski-def-1 ({-}) where
  let y be object
  func {y} → set means λit.
  for x being object holds x in it iff x = y

definition tarski-def-4 (union - [90] 90) where
  let X be set
  func union X → set means λit.
  for x being object holds
  x in it iff (ex Y being set st x in Y & Y in X)

definition tarski-def-5 ([- , -]) where
  let x be object & y be object
  func [x,y] → object equals
  {{x, y}, {x}}
```

Figure 2. Selected Definitions from Tarski in Mizar and Isabelle/Mizar

```
assume A0:A is set
let ?Q = λx. λy. (x=y & P(x))
have A1: for x,y,z being object holds
  ?Q(x, y) & ?Q(x, z) implies y = z by auto
obtain X where
A2:X is set & (for x being object holds x in X iff
  (ex y being object st y in A & ?Q(y, x)))
    using tarski-sch-1[OF A0 A1] by auto
thus ex X being set st
  (for x being object holds x in X iff x in A & P(x))
    by auto
```

The next step in the XBOOLE_0 article is the definition of the attribute empty. Our emulation of the Mizar definitions of predicates, attributes and modes is analogous to the emulation of functor definitions described above, and we skip the technical details here (see the file Mizar.thy for the exact implementation).[11] The Isabelle code used to define the attribute empty looks as follows:

```
definition xboole-0-def-1[simp]:
  let X is set attr X is empty means
    not (ex x being object st x in X)
```

And the provided definitional theorem is:

```
X is set ⟹ X is empty iff not (ex x being object st x in
X)
```

The emulation can also be used to introduce the first cluster, which is a simple theorem that states the existence of the empty set. This in turn enables us to define the empty set:

```
theorem xboole-0-cl-1[simp]:
  cluster empty for set

definition xboole-0-def-2-prefix ({}) where
  func {} → set equals the empty set
```

In a similar way we can finish the formalization of the Mizar article XBOOLE_0 in our emulated Mizar. A number of proofs can be performed completely automatically with the help of the correctly set up simplifier, introduction and elimination rules, such as for example:

```
theorem xboole-0-th-6:
  for X,Y being set st X c< Y holds
    ex x being object st x in Y & not x in X
```

For other Mizar statements longer Isar proofs are necessary, for example:

```
theorem xboole-0-th-8:
  for X,Y being set st X c< Y holds
    ex x being object st x in Y & X c= Y\{x}
```

requires a 26 lines of proof, while the original Mizar proof took 16 lines.

9. Conclusion

We have used the Isabelle logical framework to do the first steps in rather faithful emulation of the Mizar language and logic. This includes declaring and axiomatizing Mizar terms, types and related mechanisms, and providing suitable syntactic conventions corresponding to Mizar. We also provide definitional mechanisms that already quite faithfully correspond to the original Mizar syntax and semantics.

The resulting emulation has been used to develop the exact version of set theory which Mizar uses, and to show how to formalize in the Mizar environment the first two non-axiomatic articles in the MML. This includes 15 Mizar-style definitions including functors and attributes, 6 registrations (clusters) and 32 proved theorems. The total size of the development is 40kB.

9.1 Future Work

There is a lot of future work mentioned above. The mechanisms introduced are quite faithful, but occasionally they are either too strict or too liberal in comparison with Mizar. Many pieces are still missing. One obvious gap is exact emulation of the Mizar proof style, which differs in various aspects from the Isar language. We have not tried to emulate the Mizar type-inference mechanisms yet, and neither its core "by" proof checker. We also do not have mechanisms for working with the Mizar structures, automatically used properties, redefinitions, etc.

An interesting experiment which is probably not too far now is to translate the whole MML into the emulated syntax in a similar way as the TPTP and HTML exports work, and to try to import and cross-verify large parts in Isabelle. Since today's ATPs are strong enough to discharge a vast majority of the Mizar "by" steps [30], one does not really need to do the rather involved emulation of the Mizar "by" checker first. A high-level emulation

[11] The complete formalization is available at `http://cl-informatik.uibk.ac.at/cek/cpp16/`

using Isabelle automation tactics (in particular reusing parts of the Sledgehammer infrastructure [20]) will likely be sufficient for importing automatically a large number of Mizar proofs.

Acknowledgements

Kaliszyk has been supported by the Austrian Science Fund (FWF) grant P26201. Pąk has been supported by the Polish National Science Centre grant DEC-2012/07/N/ST6/02147. Urban has been supported by the ERC Consolidator grant 649043 *AI4REASON*.

References

[1] S. Agerholm and M. J. C. Gordon. Experiments with ZF set theory in HOL and isabelle. In E. T. Schubert, P. J. Windley, and J. Alves-Foss, editors, *Higher Order Logic Theorem Proving and Its Applications, 8th International Workshop*, volume 971 of *Lecture Notes in Computer Science*, pages 32–45. Springer, 1995.

[2] G. Bancerek and P. Rudnicki. A Compendium of Continuous Lattices in MIZAR. *J. Autom. Reasoning*, 29(3-4):189–224, 2002.

[3] M. Davis. Obvious logical inferences. In P. J. Hayes, editor, *IJCAI*, pages 530–531. William Kaufmann, 1981.

[4] F. Haftmann and T. Nipkow. Code generation via higher-order rewrite systems. In M. Blume, N. Kobayashi, and G. Vidal, editors, *Functional and Logic Programming, 10th International Symposium, FLOPS 2010*, volume 6009 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2010.

[5] J. Harrison. A Mizar mode for HOL. In J. von Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, volume 1125 of *Lecture Notes in Computer Science*, pages 203–220, Turku, Finland, 1996. Springer-Verlag.

[6] J. Hurd. The OpenTheory standard theory library. In M. G. Bobaru, K. Havelund, G. J. Holzmann, and R. Joshi, editors, *NASA Formal Methods*, volume 6617 of *LNCS*, pages 177–191. Springer, 2011.

[7] M. Iancu, M. Kohlhase, F. Rabe, and J. Urban. The Mizar mathematical library in OMDoc: Translation and applications. *J. Autom. Reasoning*, 50(2):191–202, 2013.

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

[9] C. Kaliszyk and A. Krauss. Scalable LCF-style proof translation. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Proc. of the 4th International Conference on Interactive Theorem Proving (ITP'13)*, volume 7998 of *LNCS*, pages 51–66. Springer, 2013.

[10] C. Kaliszyk and J. Urban. MizAR 40 for Mizar 40. *J. Autom. Reasoning*, 55(3):245–256, 2015. . URL `http://dx.doi.org/10.1007/s10817-015-9330-8`.

[11] A. Krauss and A. Schropp. A mechanized translation from higher-order logic to set theory. In M. Kaufmann and L. C. Paulson, editors, *Interactive Theorem Proving (ITP 2010)*, volume 6172 of *LNCS*, pages 323–338. Springer, 2010.

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

[13] R. Matuszewski and P. Rudnicki. Mizar: the first 30 years. *Mechanized Mathematics and Its Applications*, 4:3–24, 2005.

[14] S. Merz. Mechanizing TLA in Isabelle. In R. Rodošek, editor, *Workshop on Verification in New Orientations*, pages 54–74, Maribor, July 1995. Univ. of Maribor.

[15] A. Naumowicz. Enhanced Processing of Adjectives in Mizar. In A. Grabowski and A. Naumowicz, editors, *Computer Reconstruction of the Body of Mathematics*, volume 18(31) of *Studies in Logic, Grammar and Rhetoric*, pages 89–101. University of Białystok, 2009.

[16] S. Obua. Partizan games in Isabelle/HOLZF. In K. Barkaoui, A. Cavalcanti, and A. Cerone, editors, *Theoretical Aspects of Computing - ICTAC 2006, Third International Colloquium*, volume 4281 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2006.

[17] 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 - International Conference, CICM*, volume 9150 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 2015.

[18] L. C. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science (1990)*, pages 361–386, 1990.

[19] L. C. Paulson. Set theory for verification: I. From foundations to functions. *J. Autom. Reasoning*, 11(3):353–389, 1993.

[20] L. C. Paulson and J. C. Blanchette. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In G. Sutcliffe, S. Schulz, and E. Ternovska, editors, *The 8th International Workshop on the Implementation of Logics, IWIL 2010*, volume 2 of *EPiC Series*, pages 1–11. EasyChair, 2010.

[21] F. J. Pelletier. A brief history of natural deduction. *History and Philosophy of Logic*, 20:1 – 31, 1999.

[22] qed. The QED Manifesto. In A. Bundy, editor, *International Conference on Automated Deduction (CADE 1994)*, volume 814 of *LNCS*, pages 238–251. Springer, 1994.

[23] F. Rabe. A logical framework combining model and proof theory. *Mathematical Structures in Computer Science*, 23(5): 945–1001, 2013. . URL `http://dx.doi.org/10.1017/S0960129512000424`.

[24] P. Rudnicki. Obvious Inferences. *J. Autom. Reasoning*, 3(4):383–393, 1987.

[25] S. Schulz. E - A Brainiac Theorem Prover. *AI Commun.*, 15(2-3): 111–126, 2002.

[26] C. Schürmann. The twelf proof assistant. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009*, volume 5674 of *Lecture Notes in Computer Science*, pages 79–83. Springer, 2009.

[27] A. Tarski. Über unerreichbare Kardinalzahlen. *Fundamenta Mathematica*, 30:68–89, 1938. URL `http://matwbn.icm.edu.pl/ksiazki/fm/fm30/fm30113.pdf`.

[28] J. Urban. MoMM - fast interreduction and retrieval in large libraries of formalized mathematics. *Int. J. on Artificial Intelligence Tools*, 15 (1):109–130, 2006.

[29] J. Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning*, 37(1-2):21–43, 2006.

[30] 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. . URL `http://dx.doi.org/10.1007/s11786-008-0053-7`.

[31] M. Wenzel. Isar - A generic interpretative approach to readable formal proof documents. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99*, volume 1690 of *Lecture Notes in Computer Science*, pages 167–184. Springer, 1999.

[32] M. Wenzel. The Isabelle/Isar reference manual, 2015.

[33] 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, 21st International Conference, TPHOLs 2008*, volume 5170 of *Lecture Notes in Computer Science*, pages 33–38. Springer, 2008.

[34] F. Wiedijk. CHECKER - notes on the basic inference step in Mizar. available at http://www.cs.kun.nl/∼freek/mizar/by.dvi, 2000. URL `http://www.cs.kun.nl/~freek/mizar/by.dvi`.