Formalizing Foundations of Mathematics

Mihnea Iancu
Supervisors: Florian Rabe, Michael Kohlhase

Jacobs University Bremen

Abstract. The increase of mathematical knowledge during the last few hundred years and its development towards greater precision has led to the formalization of large parts of mathematics. There are currently several systems that allow the formalization of mathematics and that are able to formally check the correctness of mathematical proofs. However, they are based on different foundations of mathematics which means that their libraries of formal mathematics are completely independent and not translatable from one system to another. Here we give translations from Isabelle/HOL and Mizar's Tarski-Grothendieck set theory to ZFC that formalize the set theoretical semantics of Isabelle and Mizar. The translations are represented in the Edinburgh logical framework LF and include representations of ZFC, Isabelle, Isabelle/HOL, Mizar and Tarski-Grothendieck set theory in LF.

1 Introduction and Related Work

1.1 Formalized Mathematics

The 20^{th} century saw significant advances in formal set theory mostly stimulated by the discovery of paradoxes in naive set theory. The most famous paradox was found by Russell in 1901 and roughly arises from unlimited set comprehension. To conform with our intuition about logic, set theories must ensure that truth and falsity are disjoint, meaning that the theory is free of contradiction (consistent).

Despite its conceptual beauty naive set theory was not consistent and Russell's paradox, although simple, did not suggest an immediate fix. As a result several foundations were developed by mathematicians over many years but this process did not lead to a commonly accepted solution but to a continuous branching of set theories.

The two most important classes of foundations are axiomatic set theory and type theory.

Axiomatic set theory [31] assumes a universe of sets and that all mathematical objects that can be introduced are sets. A number of propositions (axioms) are chosen as fundamental truths and then (usually) first-order logic is used as a language to compose propositions about the sets and reason about them. Provided that the axioms were

chosen correctly they must not lead to contradiction and at the same time be powerful enough to generate the whole of mathematics.

Type theory [38] challenges the assumption that all objects in the universe are sets and creates a universe containing a hierarchy of types and then assigns each object to a type. In their simplest form, type theories have two classes of objects, terms and types. Intuitively, in type theory, each term x has (usually) one type A and the set theoretical reasoning is constrained to terms of the same type. This is employed by many programming languages where each variable has a certain type and functions (sum, multiplication etc.) are only valid if applied to argument of the proper type (e.g. cannot add a string to an integer).

Both set and type theory have led to many distinct foundations of mathematics. The most widely used axiomatic set theory today is **Zermelo-Fraenkel** [46,9](optionally with choice) but there are other relevant variants like **Tarski-Grothendieck** [32], von Neumann-Bernays-Gödel or Morse-Kelley [36]. For type theory, the most common today is Church's simple theory of types [7] also called **higher-order logic**(HOL) and other variants include Russel's ramified theory of types [30], dependent type theory and System F.

With the development of computer science several systems were developed with the purpose of allowing the representation of mathematical knowledge and mathematical proofs in a computer environment so that a program can check their correctness or find proofs automatically. Some of these systems were quite successful and there is a large amount of theorems that were formalized in such a way.

The **Mizar** project [35,42] started in 1973 and is "an attempt to reconstruct mathematical vernacular in a computer-oriented environment". Thus it is designed to be similar to the natural language of mathematics and part of its purpose is to provide a large library of mathematics. A major project is the translation to Mizar language of a mathematics book: "A Compendium of Continuous Lattices" [39,20]. Another reason Mizar is particularly interesting in that it is also based on untyped set theory – very similar to ZFC – and only has types implicitly through its inner syntax. This is in contrast to most other computer-based formalizations of mathematics, which are typically based on typed formalisms.

One such formalization is **Isabelle** [24], which is a generic proof assistant. Isabelle is a logical framework, and the most important logic formalized in it is HOL. Isabelle/HOL has proved quite successful and is currently the most widespread instance of Isabelle [41].

1.2 Integrating Foundations

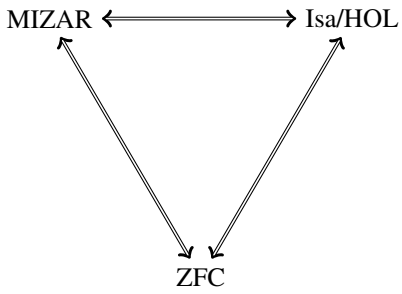
Many mathematical results and proofs are formalized in each of the above systems, but since some of them start from axiomatic set theory, and some from various type theories, these proofs are not or not easily translatable from one system to another.

Our goal is to provide a framework for such translations by using theory morphisms [8,4] which are truth preserving maps between mathematical theories. Therefore, we formalize three foundations – ZFC, Isabelle/HOL and Mizar's Tarski-Grothendieck set theory – and provide translations between them in LF. For this we are using the Twelf system which provides an implementation of the LF logical framework and also has a module system which enables the formalization of theory morphisms [28].

Isabelle/HOL and Mizar are interesting because they currently have the largest libraries of formalized mathematics while Zermelo Fraenkel is the set theory most commonly used by mathematicians. Ideally we should create morphisms between each of these theories so we can translate proofs from one to the other.

In this thesis we will describe an implementation of ZFC based on FOL and then the recovery of typed reasoning within the ZFC framework [13]. In addition we will present morphisms (views) to ZFC from other important foundations of mathematics: Isabelle/HOL and Mizar's Tarski-Grothendieck set theory. From a mathematical perspective this will provide a proof that the primitives in other typed systems can be also seen as entities within ZFC. This is also a step towards the integration of these systems with the ultimate purpose of providing a universal library of mathematics containing all the mathematical knowledge. Morphisms between theories and views - the Twelf specific implementation of this - will be explained further in Section 2

Related Work Scunak [5] is a small, stand-alone, independent system for formalizing mathematics. It is implemented in Common Lisp and is designed to support a logical framework 'just strong enough' to support natural versions of formal set theory [44]. Scunak's purpose is similar to ours since it also starts from an axiomatic set theory and attempts to provide a platform for formalizing mathematics [6].



A big success of such projects was for example the proof of the four color theorem which was fully checked by the Coq proof assistant [10]. It was the first long-standing mathematical problem to be solved using a computer program.

A similar project is Flyspeck which attempts to produce a formal proof of the Kepler conjecture [40]. It uses HOL Light which is a HOL system implemented in Ocaml [12].

The Logosphere project [26] focuses on developing a Formal Digital Library for managing and sharing mathematical knowledge. It has formalizations of HOL, Nuprl and PVS in LF and a trabslation from HOL to Nuprl [21].

There are also translations between HOL systems [19,22] as well as an interpretation of HOL-Light in Coq [15]. The OpenTheory project [14] is designed to allow proofs to be shared between the different theorem prover implementations of higher order logic, including HOL4, ProofPower/HOL and HOL Light.

A newer project is the LATIN project [17] – Logic ATlas and INtegrator – which aims at developing methods, techniques, and tools for interfacing logics and proof systems.

This thesis is organized as follows. In section 2 we will introduce LF and Twelf's module system. Section 3 will discuss the representation of ZFC in Twelf and the recovery of type theoretic semantics in untyped set theory. Section 4 will focus on the representation of the syntax of Isabelle/HOL in Twelf and the theory morphism from Isabelle to ZFC. Similarly section 5 will analyze the morphism from Mizar (using Tarski set theory) to ZFC as well as the Twelf representation of the Mizar syntax. Finally 6 will present possible future work related to this thesis.

The code that will be described here spans over several thousands of lines of code so we will only present simplified snippets when needed and rarely entire proofs as some of them are quite long. The full Twelf sources are available at [45].

Acknowledgments This thesis was done with the strong support of and in collaboration with Florian Rabe. In particular, the formalization of Isabelle/HOL described in sections 4.2 and 4.3 was done entirely by him and is presented here only as an introduction for the interpretation of Isabelle/HOL in ZFC presented in section 4.4.

2 LF

A logical framework is a meta-language used for the formalization of deductive systems.

LF [11] is a logical framework based on dependent type theory and the judgement as types methodology. It is related to Martin-Lof type theory [18] and the corner of the lambda cube [2] that extends simple type theory with dependent function types. We will work with the Twelf implementation of LF.

The declarations in Twelf are *kinded type family* symbols $a : K$ and *typed constants* $c : A$. Both can carry definitions, $c : A = t$ means that c is an abbreviation for t. The

objects of Twelf are kinds K, kinded type families $A : K$, and typed terms $t : A$. $type$ is the kind of types and $A \to type$ is the kind of type families indexed by terms of type A.

We use the Twelf notation for binding and application:

- $\Pi_{x:A} B(x)$ is $\{x : A\}\ B\ x$ (we write $A \to B$, when $x : A$ does not occur in B)
- $\lambda_{x:A} t(x)$ is $[x]\ t\ x$

We can also omit the types of bound variables if they can be inferred.

The Twelf module system [29] is based on the notions of signatures and signature morphisms. Given two signatures $Sig\ S = \{\Sigma\}$ and $Sig\ S' = \{\Sigma'\}$ a signature morphism from S to S' is a type and kind preserving map from Σ symbols to Σ' expressions.

The modular declarations are signatures and explicit morphisms called views. Signatures can be nested and may import other signatures via inclusions and structures.

Signatures	Σ	$::= \cdot \mid \Sigma,\ \texttt{sig}\ T\ =\ \{\Sigma\} \mid \Sigma,\ \texttt{view}\ v\ : S\ \to T\ = \{\sigma\}$
		$\mid\ \ \Sigma,\ \texttt{include}\ S \mid \Sigma,\ \texttt{struct}\ s\ : S\ = \{\sigma\}$
		$\mid\ \ \Sigma,\ c : A[= t] \mid \Sigma,\ a : K[= A]$
Instantiations	σ	$::= \cdot \mid \sigma,\ c := t \mid \sigma,\ a := A \mid \sigma,\ \texttt{struct}\ s := \mu$
Kinds	K	$::= \texttt{type} \mid A \to K$
Type families	A	$::= S.a^{\mu} \mid A\ t \mid \{x : A\}\ A$
Terms	t	$::= S.c^{\mu} \mid x \mid [x : A]\ t \mid t\ t$
Morphisms	μ	$::= (v \mid T.s)^*$

For constants, the notation $S.c^{\mu}$ as a constant in signature T means that c is a constant in signature S and μ is a morphism from S to T. This means that if the type of c in signature S is A then the type of $S.c^{\mu}$ in signature T is $\mu(A)$. Basically $S.c^{\mu}$ is the projection of the constant c from signature S through morphism μ.

Consider $\{\sigma : S \to T\}$ to be an explicitly given *signature morphism*. If declared on toplevel, it is called a view. Because judgments are represented as types and proofs as terms, signature morphisms must map axioms and inference rules to proofs and derived rules. Thus, views have the flavor of a theorem expressing the judgment-preserving representation of one signature in another.

Example 1 (Representation of FOL in LF).

$$
\begin{array}{l}
\texttt{sig}\ FOL = \{ \\
\quad set \quad : type. \\
\quad prop \quad : type. \\
\quad ded \quad : prop \rightarrow type. \\
\\
\quad true \quad : prop. \\
\quad false \ : prop. \\
\quad not \quad : prop \rightarrow prop. \\
\quad imp \quad : prop \rightarrow prop \rightarrow prop. \\
\quad and \quad : prop \rightarrow prop \rightarrow prop. \\
\quad or \quad : prop \rightarrow prop \rightarrow prop. \\
\quad forall : (set \rightarrow prop) \rightarrow prop. \\
\quad exists : (set \rightarrow prop) \rightarrow prop. \\
\quad eq \quad : set \rightarrow set \rightarrow prop. \\
\quad equiv \ : set \rightarrow set \rightarrow prop = [a][b](aimpb)and(bimpa). \\
\}.
\end{array}
$$

This introduces two types set and $prop$. Intuitively $prop$ stands for propositions ($true$ or $false$), set for sets and $ded\ A$ for proofs of A.

3 Zermelo Fraenkel Set Theory

3.1 Preliminaries

Zermelo Fraenkel set theory (commonly abbreviated ZFC when including axiom of choice and ZF without choice) is one of the axiomatic set theories presented in the beginning of the 20^{th} century in response to the discovery of Russel's paradox. ZFC has a single ontological notion, that of a hereditary set meaning that all elements of every set are also (hereditary) sets. In fact ZFC assumes that all objects in the domain of discourse are such sets. Its signature consists only of equality and set membership.

For the axioms of ZFC there are many equivalent formulations so for the sake of clarity we will present the one we used here:

- Axiom of extensionality
  $\forall x \forall y(\forall z(z \in x \Leftrightarrow z \in y) \Rightarrow x = y)$
- Axiom of set existence
  $\exists x\ true$
- Axiom of unordered pairing
  $\forall x \forall y \exists a(\forall z(z = x \vee z = y) \Rightarrow z \in a)$

- Axiom of union
  $\forall x \exists a \forall z (\exists x (x \in X \wedge z \in X) \Rightarrow z \in a)$
- Axiom of power set
  $\forall x \exists a \forall z ((\forall t (t \in z \Rightarrow t \in x)) \Rightarrow z \in a)$
- Axiom schema of specification
  $\forall X \exists a (\forall z ((z \in X \wedge (\varphi\ z)) \Leftrightarrow z \in a))$ for a unary predicate φ (possibly containing free variables).
- Axiom schema of replacement :
  $\forall a (\forall x (x \in a) \Rightarrow \exists^! y (\varphi\ x\ y)) \Rightarrow \exists b (\forall y (\exists x (x \in a \wedge \varphi\ x\ y)) \Leftrightarrow y \in b)$ for a binary predicate φ (possibly containing free variables).
- Axiom of regularity
  $\forall x (\exists t (t \in x)) \Rightarrow (\exists y (y \in x \wedge not (\exists z (z \in x \wedge z \in y))))$.
- Axiom of choice
  $\forall a ((\forall x (x \in a) \Rightarrow \forall y (y \in a) \Rightarrow (x = y \vee not (\exists z (z \in x \wedge z \in y)))) \wedge (\forall x (x \in a) \Rightarrow (\exists t (t \in x)))) \Rightarrow (\exists c (\forall x (x \in a) \Rightarrow (\exists^! y (y \in c \wedge y \in x))))$.
- Axiom of Infinity
  $\exists x (\emptyset \in x \wedge \forall y (y \in x \Rightarrow (y \cup \{y\}) \in x))$

Note that there no terms except for the variables. Concrete sets and operations on them are introduced as abbreviations after proving their unique existence. This kind of abbreviations cannot be formalized in FOL, which is why we will introduce a description operator below.

3.2 Untyped Set Theory

In our case, the First Order Logic and the axioms of ZFC are already defined, and provide a backbone for building the untyped and typed reasoning. For the sake of clarity, some of the Twelf declarations in this paper are slightly simplified to make it easier to understand. The rest of the Twelf listings presented in the here will thus be consistent with the simple FOL presented in section 2.

For example, Axiom of extensionality can be written in Twelf as:

$$ax\_extensionality : ded\ forall\ [x]\ forall\ [y] \\ (forall\ ([z]\ z\ in\ x\ equiv\ z\ in\ y) imp\ x\ eq\ y).$$

Starting from FOL and the axioms of ZFC we can define all notions of set theory. On this basis we can build the untyped set theory by defining union, pair, cartesian product, functions and even natural numbers.

This is in contrast to Mizar where primitive function symbols are used for singleton, unordered pair, and union [35,34], and to Isabelle/ZF where primitive function symbols

are used for empty set, powerset, union, infinite set, and replacement [25]. Since we are using this particular axiomatization we have no primitive function symbols which means that there are no terms except variables. Therefore, we add the (definite) description operator $\delta : \{F : set \to prop\} \vdash \exists^!([x]\, F\ x) \to set$, which takes a formula $F(x)$ with a free variable x and a proof of $\exists^! x.F(x)$ and returns a set (where the binder $\exists^!$ means unique existence and is defined as an abbreviation). Because a proof is passed as an argument, the LF type system guarantees that δ encodes the accepted mathematical practice of giving a name for a uniquely determined object. δ is axiomatized using an axiom scheme $F(\delta\ F\ P)$ (from which we can derive proof irrelevance)

Using this operator for implicit definitions (which we name $imlpDef$ in our LF signature) we can introduce unordered pair, big union, powerset and other function symbols. Firstly we introduce two more theorems:

- $spec\_unique$: if a class $\varphi : set \to prop$ defines a set then that set exists uniquely
- $shrink$: if there is a set X larger then the class $\varphi : set \to prop$ then that class is a set x.

We won't present the proofs here as they are quite long but they roughly result from axiom of extensionality and axiom schema of specification.

$$\begin{aligned} spec\_unique : &\ \ ded\ (exists\ [x]\ forall\ ([z]\ (Phi\ z)\ equiv\ z\ in\ x)) \\ &\to ded\ existsU\ [x]\ forall\ ([z]\ (Phi\ z)\ equiv\ z\ in\ x) = \cdots \\ shrink\ \ \ : &\ \ ded\ (exists\ [X]\ forall\ ([z]\ (Phi\ z)\ imp\ z\ in\ X)) \\ &\to ded\ (exists\ [x]\ forall\ ([z]\ (Phi\ z)\ equiv\ z\ in\ x)) = \cdots \end{aligned}$$

Now we can define, for example, unordered pair in the following way:

$$\begin{aligned} is\_uopair : &\ set \to set \to set \to prop \\ &= [x][y][a](forall\ [z]\ (z\ eq\ x\ or\ z\ eq\ y)\ equiv\ z\ in\ a). \\ p\_uopair\ : &\ ded\ existsU\ (is\_uopair\ A\ B) \\ &= spec\_unique(shrink\ (forall2E\ ax\_pairing\ A\ B)). \\ uopair\ \ \ : &\ set \to set \to set \\ &= [a][b]\ implDef\ (is\_uopair\ a\ b)\ p\_uopair. \end{aligned}$$

Union of two sets $A\ B$ which is defined as the bigunion of the unordered pair of A and B.

$$\begin{aligned} union : &\ set \to set \to set \\ &= [x][y]\ bigunion\ (uopair\ x\ y). \end{aligned}$$

(Ordered) Pair of two sets A and B is defined as the set $\{\{A\}, \{\{B\}, \emptyset\}\}$ Note that we are not using the common Kuratowski pair: $\{\{a, b\}, \{a\}\}$ because reasoning about Kuratowski pairs requires the law of excluded middle and even though we assume that, we avoid using it if possible to make our development more reusable.

$$
\begin{aligned}
pair :\ & set \to set \to set \\
& = [a][b]\ uopair\ (singleton\ a) \\
& \quad (uopair\ (singleton\ b)\ empty).
\end{aligned}
$$

Product of two sets A and B is defined as the set containing $pair\ x\ y$ for every $x \in A$ and $y \in B$.

$$
\begin{aligned}
prod :\ & set \to set \to set \\
& = [a][b]\ bigunion \\
& \quad (image\ ([x]\ image\ ([y]\ pair\ x\ y)\ b)\ a).
\end{aligned}
$$

We can define the set of all relations between two sets A and B as the powerset of the product of A and B:

$$rel : set \to set \to set = [a][b]\ (powerset\ (a\ prod\ b)).$$

Now we can define a function $functional$ that takes 3 sets as arguments: A, B and R and returns true if and only if R is a relation on $A\ B$ and for every $x \in A$ there is at most one $y \in B$ such that $pair\ x\ y$ is in R.

$$
\begin{aligned}
functional :\ & set \to set \to set \to prop \\
& = [a][b][r]\ (r\ in\ rel\ a\ b)\ and \\
& \quad (forall\ [x]\ forall\ [y]\ forall\ [y'] \\
& \quad ((pair\ x\ y)\ in\ r\ and\ (pair\ x\ y')\ in\ r)\ imp\ y\ eq\ y').
\end{aligned}
$$

We can now proceed to defining functions but for that we need another two notions; $image$ and $filter$. For the sake of simplicity we won't present the Twelf definitions for them here but just their mathematical meaning:

- $filter\ X\ \varphi = \{x : x \in X \wedge \varphi(x)\}$ where X is a set and $\varphi : set \to prop$ is a unary predicate
- $image\ F\ A = \{F(x) : x \in A\}$ where A is a set and $F : set \to set$ is a unary function

Now we define partial functions as relations that are functional and total functions as partial functions that are also defined in every point.

$$
\begin{aligned}
&pfunc : set \to set \to set = [a][b]\ filter\ (rel\ a\ b)([r]\ functional\ a\ b\ r). \\
&func\ \ : set \to set \to set = [a][b]\ filter\ (pfunc\ a\ b) \\
&\qquad\qquad\qquad\qquad ([r]\ forall\ [x]\ x\ in\ a\ imp\ exists\ [y]\ (pair\ x\ y)\ in\ r).
\end{aligned}
$$

Finally, we can continue with $lambda$ and $apply$:

- $lambda\ a\ f = \{(x, f(x)) : x \in a\}$

– $apply\ f\ a = f(a)$ value unspecified unless $definedAt\ f\ a$

$$lambda : set \rightarrow (set \rightarrow set) \rightarrow set = [a][f]\ image\ ([x]\ pair\ x\ (f\ x))\ a.$$
$$apply\ \ : set \rightarrow set \rightarrow set = [f][a]\ bigunion\ (img\ f\ a).$$

Since we are starting from just the axioms, we can thus infer the validity of all the concepts defined and type-check all the proofs and definitions. Consequently, every such definition is accompanied by functions that capture their properties. We prove the usual conversion rules for pairs and functions (β, η).

For example for $pair$ we have $pi1$ (and $pi2$) which, when applied to an ordered pair, returns the first (and second) element of the pair. For each of these we need to prove that the properties we associate with it hold. For example we need to prove that $pi1\ (pair\ X\ Y)$ is X.

This means that the definitions contain more than the syntax of the functions defined but also allow us to recover their meaning.

3.3 Typed Set Theory

In Computer Science, for practical reasons typed set theory is much more appealing than untyped. Types are needed for efficiency and decidability and the systems that use typing like Mizar and HOL are very successful. We define typed set theory by first defining classes. They are represented in LF using proof-carrying terms, pairs of a set and a proof that the set is in the class.

$$Class\ \ : (set \rightarrow prop) \rightarrow type.$$
$$celem\ \ : \{a : set\}\ ded\ P\ a \rightarrow Class\ P.$$
$$cwhich : Class\ P \rightarrow set.$$
$$cwhy\ \ : \{a : Class\ P\}\ ded\ (P\ (cwhich\ a)).$$

$$Elem\ \ : set \rightarrow type = [a]\ Class\ [x]\ x\ in\ a.$$
$$elem\ \ : \{a : set\}\ ded\ a\ in\ A \rightarrow Elem\ A = [a][p]\ celem\ a\ p.$$
$$which\ \ : Elem\ A \rightarrow set = [a]\ cwhich\ a.$$
$$why\ \ : \{a : Elem\ A\}\ ded\ (which\ a)\ in\ A = [a]\ cwhy\ a.$$

$Class\ P$ simulates $\Sigma_{x:i}\ (ded\ P\ x)$; From this perspective, $celem$ corresponds to *pair*, $cwhich$ to *pi1* and $cwhy$ to *pi2*. $Elem\ A$ specializes $Class$ to those classes that are sets and thus gives the intuitive symbols specific to typed set theory.

Then :

$X : Elem\ A$

means that X contains an element x and a proof that x is in A. This requires us to redefine the concepts from untyped set theory but we can use those definitions here.

For example, the equality of two Elems is defined as the equality of the elements they encode :

$Eq : ElemA \to ElemA \to prop = [a][b]\ (which\ a)\ eq\ (which\ b)$.

Now Eq takes two arguments which are both $Elem\ A$ so it behaves it a way consistent with typed set theory, it can only be applied to arguments of the same type. By raising the other operations from untyped to this level using $Elem$s we can define all the other functions of typed set theory within ZFC and prove their properties, thus allowing for typed reasoning within untyped set theory.

We can consider $func\ A\ B$ as the set of functions from A to B and then define $Lambda$, App, $Forall$, $Filter$, Eq and $ifte$.

$$
\begin{array}{ll}
func & : set \to set \to set = \ldots \\
Lambda & : (Elem\ A \to\ Elem\ B) \to Elem(func\ A\ B) \\
 & = [F][x]\ elem\ (apply\ (which\ F)(which\ x))(funcE\ (why\ F)(why\ x)). \\
App & : Elem(func\ A\ B) \to Elem\ A \to Elem\ B = \ldots \\
Forall & : (Elem\ A \to prop) \to prop = \ldots \\
Filter & : (Elem\ A \to prop) \to set = \ldots \\
Eq & : Elem\ A \to Elem\ A \to prop \\
 & = [a][b]\ (which\ a)\ eq\ (which\ b) \\
ifte & : \{F : o\}(\vdash\ F \to Elem\ A) \to (\vdash\ \neg F \to Elem\ A) \to Elem\ A = \ldots
\end{array}
$$

4 Isabelle/HOL

4.1 Preliminaries

Isabelle is a logical framework and a generic interactive theorem prover based on polymorphic higher order logic [37]. It is a grown and widely used system, which has led to a rich ontology of Isabelle declarations.

In this paper we will only consider the core and module system declarations and even among those we will focus on a proper subset of Isabelle's power.

The inner syntax of for *terms*, *types*, *prop*ositions and *proof* terms – called the Pure language – is given by an intuitionistic higher order logic with shallow polymorphism.

For the full syntax of Isabelle we refer to [37].

4.2 Formalizing Isabelle

The representation of Isabelle in LF proceeds in two steps. In a first step, we declare an LF signature $Pure$ for the inner syntax of Isabelle. This syntax declares symbols for all primitives that can occur (explicitly or implicitly) in $Pure$ expressions. In a second

step, every Isabelle expression E is represented as an LF expression $\ulcorner E \urcorner$. Finally we have to justify the adequacy of the encoding. In this section we will only sketch the definition of $\ulcorner E \urcorner$. A full definition is given in the appendix.

$$
\begin{array}{lll}
\texttt{sig}\ Pure = \{ & & \\
\quad tp & : \texttt{type}. & \\
\quad \Rightarrow & : tp \to tp \to tp. & \texttt{infix}\ \text{right}\ 0\ \Rightarrow. \\
\quad tm & : tp \to \texttt{type}. & \texttt{prefix}\ 0\ tm. \\
\quad \lambda & : (tm\ A \to tm\ B) \to tm\ (A \Rightarrow B). & \\
\quad @ & : tm\ (A \Rightarrow B) \to tm\ A \to tm\ B. & \texttt{infix}\ \text{left}\ 1000\ @. \\
 & & \\
\quad prop & : tp. & \\
\quad \bigwedge & : (tm\ A \to tm\ prop) \to tm\ prop. & \\
\quad \Longrightarrow & : tm\ prop \to tm\ prop \to tm\ prop. & \texttt{infix}\ \text{right}\ 1\ \Longrightarrow. \\
\quad \equiv & : tm\ A \to tm\ A \to tm\ prop. & \texttt{infix}\ \text{none}\ 2\ \equiv. \\
 & & \\
\quad \vdash & : tm\ prop \to \texttt{type}. & \texttt{prefix}\ 0\ \vdash. \\
\quad \bigwedge\text{I} & : (x : tm\ A \vdash (B\ x)) \to \vdash \bigwedge([x]B\ x). & \\
\quad \bigwedge\text{E} & : \vdash \bigwedge([x]B\ x) \to \{x : tm\ A\} \vdash (B\ x). & \\
\quad \Longrightarrow\text{I} & : (\vdash A \to \vdash B) \to \vdash A \Longrightarrow B. & \\
\quad \Longrightarrow\text{E} & : \vdash A \Longrightarrow B \to \vdash A \to \vdash B. & \\
\quad \text{refl} & : \vdash X \equiv X. & \\
\quad \text{subs} & : \{F : tm\ A \to tm\ B\} \vdash X \equiv Y \to \vdash F\ X \equiv F\ Y. & \\
\quad \text{exten} & : \{x : tm\ A\} \vdash (F\ x) \equiv (G\ x) \to \vdash \lambda F \equiv \lambda G. & \\
\quad \text{beta} & : \vdash (\lambda[x : tm\ A]F\ x)\ @\ X \equiv F\ X. & \\
\quad \text{eta} & : \vdash \lambda\ ([x : tm\ A]F\ @\ x) \equiv F. & \\
\}. & &
\end{array}
$$

Fig. 1. LF Signature for Isabelle

For the *inner syntax*, the LF signature $Pure$ is given in Fig. 1. This is a rather typical encoding of higher-order logic in LF. Pure types τ are encoded as LF-terms $\ulcorner \tau \urcorner : tp$ and Pure terms $t :: \tau$ as LF-terms $\ulcorner t \urcorner : tm\ \ulcorner \tau \urcorner$. Note that contrary to the encoding of HOL in Isabelle, the LF function space $A \to B$ with λ-abstraction $[x : A]\, t$ and application $f\ t$ is distinguished from the encoding $tm\ (\ulcorner \sigma \urcorner \Rightarrow \ulcorner \tau \urcorner)$ of the Isabelle function space with application $\ulcorner f \urcorner @ \ulcorner t \urcorner$ and λ-abstraction $\lambda([x : tm\ \ulcorner \tau \urcorner] \ulcorner t \urcorner)$. Pure propositions φ are encoded as LF-terms $\ulcorner \varphi \urcorner : prop$, and derivations of φ as LF-terms of type $\vdash\ \ulcorner \varphi \urcorner$. Where possible, we use the same symbol names in LF as in Isabelle, and we can also mimic most of the Isabelle operator fixities and precedences.

The signature $Pure$ only encodes how composed Pure expressions are formed from the atomic ones. The atomic expressions – variables and constants etc. – are added when encoding the outer syntax as LF declarations. For the *non-modular declarations*, this is straightforward, and overview is given in the following table:

Expression	Isabelle	LF
base type, type operator	$(\alpha_1, \ldots, \alpha_n)\ t$	$t : tp \to \ldots \to tp \to tp$
type variable	α	$\alpha : tp$
constant	$c :: \tau$	$c : tm\ \ulcorner\tau\urcorner$
variable	$x :: \tau$	$x : tm\ \ulcorner\tau\urcorner$
assumption/axiom/definition	$a : \varphi$	$a : \vdash \ulcorner\varphi\urcorner$
theorem	$a : \varphi\ P$	$a : \vdash \ulcorner\varphi\urcorner = \ulcorner P\urcorner$

For a proof of the adequacy of this formalization we refer to [27].

4.3 Formalizing HOL

The fragment arising from translating only the primitive declarations of HOL is given in the upper part of Fig. 2. For readability, we have added the auxiliary functions $\uparrow$, $\doteq'$, and $\longrightarrow$. These have the effect that, e.g., $A \longrightarrow B$ in Isabelle can be encoded as $A \longrightarrow B$ in LF, which abbreviates $\longrightarrow' Pure.@\, A\, Pure.@\, B$. We also use an `open` declaration to use some Pure symbols without qualification. (Note that the defined constants $true$, $false$, and disjunction $|$ are not expanded in the axiom of excluded middle.)

We refer to [27] for the details of the Formalization of Isabelle/HOL in LF.

4.4 Interpreting Isabelle/HOL in ZFC

We interpret Isabelle in ZFC by creating a view from our Isabelle signatures ($Pure$ and HOL to ZFC). The basic idea is that we map Isabelle types as non-empty sets and Isabelle propositions as booleans. Thus $\vdash x$ can be interpreted in ZFC as $ded\ x\ eq\ 1$

Isabelle	ZFC
$a : tp$	$PureSem(a) : Class$ *non-empty.*
$t : a$	$PureSem(t) : Elem\ (cwhich\ PureSem(a))$.
$p : ded\ F$	$PureSem(p) : ded\ PureSem(F)$.

The rest of the $Pure$ signature is interpreted in a straightforward way as presented in Fig. 4 The mapping of Isabelle propositions to booleans does not seem natural as they seem the equivalent of propositions in set theory. However Isabelle considers $prop$ to be a regular type which means that it should be interpreted as a set. For this reason

$\texttt{sig}\ HOL = \{$
$\quad \texttt{include}\ Pure\ \texttt{open}\ tp\ tm \vdash \Rightarrow prop\ \lambda\ @\ \bigwedge \Longrightarrow \equiv.$
$\quad bool : tp.$
$\quad trueprop : tm\ bool \Rightarrow prop.$
$\quad \uparrow : tm\ bool \to tm\ prop$
$\quad = [x]\, trueprop\, @\, x.$ $\qquad \texttt{prefix}\ 3\ \uparrow.$
$\quad the : tm\ (A \Rightarrow bool) \Rightarrow A.$
$\quad eps : tm\ (A \Rightarrow bool) \Rightarrow A.$
$\quad \doteq' : tm\ A \Rightarrow A \Rightarrow bool.$
$\quad \doteq : tm\ A \to tmA \to tm\ bool$
$\quad = [x][y] \doteq'\ @\, x\, @\, y.$ $\qquad \texttt{infix}$ left 50 $\doteq.$
$\quad \longrightarrow' : tm\ bool \Rightarrow bool \Rightarrow bool.$
$\quad \longrightarrow : tm\ bool \to tm\ bool \to tm\ bool$
$\quad = [x][y] \longrightarrow'\ @\, x\, @\, y.$ $\qquad \texttt{infix}$ left 25 $\longrightarrow.$
$\quad refl : \vdash\uparrow X \doteq X.$
$\quad subst : \vdash\uparrow S \doteq T \longrightarrow P\, @\, S \doteq P\, @\, T.$
$\quad ext : \vdash (\bigwedge[x : tm\ A] \uparrow F\, @\, x \doteq G\, @\, x) \Longrightarrow\uparrow (\lambda[x]\, F\, @\, x) \doteq (\lambda[x]\, G\, @\, x).$
$\quad the\_eq\_trivial : \vdash\uparrow the\, @\, (\lambda[x]\, x \doteq A) \doteq A.$
$\quad someI : \vdash\uparrow\ (P\, @\, X) \longrightarrow\uparrow (P@(eps@P)).$
$\quad impI : \vdash (\uparrow P \Longrightarrow\uparrow Q) \Longrightarrow\uparrow (P \longrightarrow Q).$
$\quad mp : \vdash\uparrow (P \longrightarrow Q) \Longrightarrow\uparrow P \Longrightarrow\uparrow Q.$
$\quad iff : \vdash\uparrow (P \longrightarrow Q) \longrightarrow (Q \longrightarrow P) \longrightarrow (P \doteq Q).$
$\quad true\_or\_false : \vdash\uparrow (P \doteq true)\,|\,(P \doteq false).$
$\}.$

Fig. 2. LF signature for HOL

we use booleans which are an LF encoding of the set $\{0, 1\}$ where $1 = \{0\}$. The fact that Isabelle types map to non-empty sets means we need to write definitions in ZFC for functions between non-empty sets. In other words for each function that we need for the Isabelle interpretation we have to prove that when taking non-empty sets as arguments it also returns non-empty sets. For that we create another LF signature which includes ZFC and where we defined the extra operators needed just for this translation in order to simplify the view. First we need a specific type for non-empty sets:

$$neset : type = Class\ nonempty.$$
$$elems : neset \to type = [a]\ Elem\ (cwhich\ a).$$

Next we can define function only for this type using the ones we have for all sets. For example for $lambda$ and $apply$ we have :

$$
\begin{array}{l}
lam: \ (elems\ A \to elems\ B) \to elems(fun \to A \to B) \\
\quad = [f]\ elem\ (which\ (Lambda\ f)) \\
\quad\quad (congP\ (sym\ ceq\_which)\ ([x]\ (which\ (Lambda f))\ in\ x)(why\ (Lambda\ f))). \\
app\ :\ elems\ (fun\ A\ B) \to elems A \to elems\ B \\
\quad = [f][a]\ elem\ (which\ (Apply\ (cast\ (subset\_eq\ subset\_refl\ ceq\_which)\ f)\ a)) \\
\quad\quad (why\ ((Apply\ (cast\ (subset\_eq\ subset\_refl\ ceq\_which)\ f)\ a))).
\end{array}
$$

$$
\begin{array}{l}
\texttt{view}\ PureSem : Pure \to ZFC = \{ \\
\quad tp \quad := neset. \\
\quad \Rightarrow \quad := func. \\
\quad tm \quad := elem. \\
\quad \lambda \quad := lambda. \\
\quad @ \quad := app. \\
\quad prop := bool. \\
\quad \bigwedge \quad := \forall. \\
\quad \Longrightarrow \ := \Rightarrow. \\
\quad \equiv \quad := eq. \\
\quad \vdash \quad := [x] \vdash\ eq\ x\ 1. \\
\quad \vdots \\
\}.
\end{array}
$$

Fig. 4. $Pure \to ZFC$ View

Similarly, we obtain a view from HOL to ZFC. Again we only give a fragment: Note that Isabelle's *bool* is also mapped to $booleans$. The mapping for the basic operators is shown in Fig. 5

For the full encoding we refer to [17].

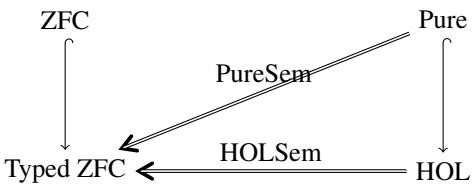


Fig. 3. Isabelle/HOL → ZFC

The mapping we described is shown if Fig. 3 where double arrows represent views.

$$
\begin{array}{l}
\texttt{view}\ HOLSem : HOL \to ZFC = \{ \\
\quad bool \quad := bool. \\
\quad trueprop := [x]\, x. \\
\quad the \quad := [f : Elem\ (func\ A\ bool)]\, ifte\ (\exists^{!}[x]\, x \in (filter\ f)) \\
\qquad\qquad ([p]\, (choice\ (elem\ (filter\ f)(\Rightarrow_{Elim}\ \ P\ p)))) \\
\qquad\qquad ([p]\, choice\ A). \\
\quad the \quad := [f : Elem\ (func\ A\ bool)]\, ifte\ (nonempty\ (filter\ f)) \\
\qquad\qquad ([p]\, (choice\ (elem\ (filter\ f)\ p))) \\
\qquad\qquad ([p]\, choice\ A). \\
\quad \doteq' \quad := Eq. \\
\quad \vdots \\
\quad typedef \ := [A]\, [f : Elem\ (func\ A\ bool)]\, [p]\, elem\ (filter\ f)p. \\
\quad \vdots \\
\}.
\end{array}
$$

Fig. 5. $HOL \to ZFC$ View

5 Mizar

5.1 Preliminaries

Mizar is a language designed in the seventies with the purpose of representing mathematical proofs in the computer. The proof checker is written in Pascal and the Mizar language is very similar to that of formal written mathematics which makes the system all the more impressive. Besides the system itself a major achievement of the Mizar project is the MML (Mizar Mathematical Library) which currently contains over 50000 theorems and 9500 definitions [43]. Because the Mizar language is so similar to the mathematical vernacular, the articles in the MML can be translated to LaTeX automatically while also remaining close enough to the mathematical language to be understood by people that are not familiar with Mizar. There is even a *Journal of Formalized Mathematics* which is quite consistent and made up entirely of such translated Mizar articles.

Mizar is based Tarski-Grothendieck set theory which is more general than ZFC due to it's Tarski axiom [33] which implies the axioms of infinity, choice and powerset but also implies the existence of inaccessible cardinals which makes the ontology of Tarski-Grothendieck set theory richer. This is relevant when creating a view from Mizar to ZFC because the Tarski Axiom cannot be defined within ZFC. However the fact that TG is richer than ZFC means that we can create a view from ZFC to Mizar and since we have one from Isabelle to ZFC we can infer the view from Isabelle to Mizar.

Mizar proof checker is based on classical logic which thus implies the law of excluded middle, De Morgan's laws and double negative elimination. Mizar includes by

default a type any to which any other type expands. It also allows the definition of new types. The basic ones are defined in the `Hidden` article which is automatically included in the every Mizar article.

The basic types in Mizar are:Any, set, $Element\ of\ X$,$DOMAIN$, $Subset\ of D$, $SUBDOMAIN\ \ of D$, $Real$ and Nat. Here X denotes a set and D a domain. The type any is the broadest type in Mizar and every other types expands to it. set is the set of all sets and is identical to any. $DOMAIN$ is the set of all non-empty sets. $SUBDOMAIN of D$ denotes all subsets of a domain D that are also domains (non-epmty). $Real$ and Nat are real and natural numbers respectively.

One of Mizar's most interesting parts is the Mizar attributes and mode. This allows the definition of new types. The definition of an attribute has the following form:

definition let x_1 be ϑ_1, ... , let x_n be ϑ_n;

attr Δ -> $\vartheta(x_1, ..., x_n)$ means $\delta(x_1, ..., x_n, it)$;

end;

In the above Δ is the symbol of the attribute and the types $\vartheta_1, ..., \vartheta_n$ are called parameters of the attribute Δ. it is a placeholder for the result of $\vartheta(x_1, ..., x_n)$. Then we can say τ is Δ meaning that $\delta(x_1, ..., x_n, \tau)$. For example $non - empty$ is an attribute of the type set.

This means we can use mode to get a new type:

definition

mode $DOMAIN$ is non-empty set;

end;

Thus, $DOMAIN$ is the type containing all non-empty sets, meaning it's a subtype of set and can be expanded to it. This is how Mizar introduces $DOMAIN$'s (and also how we do it here).

5.2 Formalizing Mizar

Our formalization of Mizar starts with separating any and set according to their intuitive meaning. In Mizar the difference between them is rather blurry but their intent seems rather clear. any is the type that encompasses all the elements in the universe of discourse and its primary purpose is that any type can be expanded to it. set is simply the type of all sets which happens to coincide with the type any because the universe of discourse of the set theory used by Mizar is made entirely of sets. In our implementation any and set are different types but, as in Mizar, they are isomorphic. Thus we create a type any and a different type tp for Mizar types like set or $DOMAIN$. We recover Mizar typing using terms only we use be instead of the more traditional tm to mimic Mizar's syntax. We define :

$be : tp \rightarrow type$

And then we can write variable introductions in a similar way to Mizar as shown in Fig. 6

Expression	Mizar	LF
type variable	α	$\alpha : tp$
constant	$let\ c\ be\ \tau$	$c : be\ \ulcorner\tau\urcorner$
variable	$let\ x\ be\ \tau$	$x : be\ \ulcorner\tau\urcorner$
assumption/axiom/definition	$definition\ \varphi\ end;$	$a :\ proof \ulcorner\varphi\urcorner$
theorem	$theorem : \varphi\ proof\ P\ end;$	$a : \vdash \ulcorner\varphi\urcorner = \ulcorner P\urcorner$

Fig. 6. Mizar-LF type equivalences

The Twelf declaration of the Mizar formal logic is:

Thus $prop$, tp and any are default types. prop corresponds to it's FOL equivalent, any represents the Mizar type and tp is the type that allows the declaration of new types. For example set and $DOMAIN$ will be introduced as being of type tp. be is the way of using domains for sets. When introducing a new variable (for example of type set) Mizar uses: $Let\ x\ be\ set$ In our declaration be is a function that takes a tp and returns a type. So we can introduce a variable of type set as : $\{x : be\ set\}$ $func\_def$ is used to define functions. It takes a function $F : be\ A \rightarrow prop$ and a proof of correctness (existence and uniqueness) and returns the (unique) $be\ A$ that satisfies F. We can use that to define functions, for example for unordered pair:

$$\begin{aligned} uopair : set \rightarrow set \rightarrow set = {} & [x][y]\ func\_def\ ([it]\ for\ [z : be\ set] \\ & (z\ in\ it)\ iff\ (t\ eq\ x\ or\ t\ eq\ y)) \\ & (uopair\_existence)\ (uopair\_uniqueness) \end{aligned}$$

Another interesting particularity of Mizar's formal logic is that they don't consider or and imp as axioms but they define them using and and not as shown in Fig. 7. In this sense they use the standard way of defining them, or as $\neg((\neg a) \wedge (\neg b))$ and imp as $\neg(a \wedge (\neg b))$. This implies that they are using classical logic since the law of excluded middle can be immediately inferred:

$$\begin{aligned} tnd : proof\ A\ or\ (not\ A) = {} & proof\_by\_contradiction\ ([p]\ notE \\ & (andEr\ (proof\_by\_contradiction\ ([q]\ notE\ p\ q))) \\ & (andEl\ (proof\_by\_contradiction\ ([q]\ notE\ p\ q)))). \end{aligned}$$

```
sig Mizar_FOL = {
  prop      : type.
  tp        : type.
  any       : type.
  sub       : tp → tp → type.
  ;         : any → tp → type.                                   infix right 30 ;.
  be        : tp → type.
  be!       : {x : any} x; A → be A.
  proof     : prop → type.                                       prefix 0 proof.
  ⋮
  or        : prop → prop → prop = [a][b] not((not a) and (not b)).
  imp       : prop → prop → prop = [a][b] not(a and (not b)).
  ⋮
  func_def  : {f : be A → prop}(proof ex [x] f x) →
              (proof for [x] for [y] (f x and f y) implies (x eq y)) → be A.
  func_prop : {F}{EX}{UNQ} proof F (func_def F EX UNQ).
  attr      : tp → type = [a] (be a → prop).
  is        : be A → attr A → prop = [x][a] a x.
  mode      : {a : tp} attr a → tp.
  ⋮
}.
```

Fig. 7. LF signature for Mizar

```
sig HIDDEN = {
  include MIZ_FOL open.
  set                : tp.
  in                 : be set → be set → prop.                  infix right 35 in.
  non − empty        : attr set = ([y] ex [x] x in y).
  Element_of         : be set → tp.
  DOMAIN             : tp = mode set non − empty.
  Subset_of          : be DOMAIN → tp.
  SUBDOMAIN_of       : be DOMAIN → tp = [x] mode (Subset_of x) non − empty.
  Real               : tp.
  Nat                : tp.
  set_ax             : X; set.
  DOMAIN_ax1         : (X; DOMAIN) iff (nonempty X).
  DOMAIN_ax2         : (X; DOMAIN) iff (nonempty X).
}.
```

Fig. 8. LF signature for the HIDDEN article

The $Hidden$ signature introduces the types described above. For example set is a tp and $DOMAIN$ is defined as $mode\ set\ non-empty$ which is the type of all non-empty sets.

5.3 Formalizing Tarski set theory

```
sig Mizar_Tarski = {
   include HIDDEN open.
   ⋮
   fraenkel        : {A : be set} P : be set → be set → prop
                     proof (for [x : be set] for [y : be set] for [z : be set]
                     ((P x y) and (P x z)) implies (y eq z))
                     → proof (ex [X] for [x] ((x in X) iff (ex [y] y in A and (P y x)))).
   pair            : be set- > be set- > be set = [x][y] uopair (uopair x y) (singleton x).
   isomorphic      : be set → be set → prop = [x][y] ex [z] (
                     for [a] (a in x implies ex [b] b in y and (pair a b) in z) and
                     for [b] (b in y implies ex [a] a in x and (pair a b) in z) and
                     for [a] (for [b] (for [c] ( for [d](
                     (pair a b) in Z and (pair c d ) in Z implies (a eq c) and (b eq d)
                     ))))).
   subset_closed   : {m} prop = [m] for [x] (for [y] ((( x in m) and (y c= x)) implies (y in m))).
   powerset_closed : {m} prop = [m] for [x] (x in m implies (ex [z] z in m and
                     (for [y] y c= x implies y in z))).
   tarski_axiom    : proof for [n] (ex [m] (
                     n in m and
                     subset_closed m and
                     powerset_closed m and
                     for [x] (x c= m implie ((isomorphic x m) or x in m)))).
   ⋮
}.
```

Fig. 9. LF signature for Tarski set theory

Tarski set theory is defined in the standard way using Mizar formal logic described in section 5.2 Since we are planning in creating a view from Tarski to ZFC we only need to define the axioms and then the other concepts will be inferred from the view. This means that the declaration of Tarski set theory in Twelf is merely a translation of the Tarski article [34] from the Mizar Mathematical Library [43] to the Twelf system. The definitions for $singleton$, $uopair$ and $union$ are inferred with $func\_prop$ and their

existence and uniqueness are considered axioms. This is formally the same as done in Mizar, except Mizar offers the possibility to suppress the request for proofs when defining axioms, while in Twelf the existence and uniqueness proofs themselves must be given as axioms.

5.4 Interpreting Tarski in ZFC

Having defined the LF signatures for Mizar base, article hidden and Tarski set theory we can give the semantics of Mizar as three LF signature morphisms from MIZ_FL, HIDDEN and MIZ_TARSKI to ZFC. The basic idea if this is interpreting Mizar domains (tp) as unary predicates that are $true$ for sets that belong to the domain. For example the predicate $p1(x) = true$ defines sets while the predicate $p2(x) = \exists y(y \in x)$ defines non-empty sets, or Mizar's DOMAIN. This means be maps to $[f]\ Class\ f$.

Mizar_LF	ZFC
$\alpha : prop$	$MizSem(\alpha) : prop$
$\alpha : any$	$MizSem(\alpha) : set$
$\tau : tp$	$MizSem(\tau) : set \to prop$
$\alpha : be\ \tau$	$MizSem(\alpha) : Class\ MizSem(\tau)$
$a : proof\ P$	$MizSem(a) : ded\ MizSem(P)$

$\texttt{view}\ MizSem : MIZ\_FOL \to ZFC = \{$
$\quad prop := prop.$
$\quad any := set.$
$\quad tp := set \to prop.$
$\quad sub := [f][g]\ ded\ forall\ [x]\ (f\ x)\ imp\ (g\ x).$
$\quad ; := [a][f]\ ded\ f\ a.$
$\quad be := [f]\ Class\ f.$
$\quad be! := [A][x][p]\ celem\ x\ p.$
$\quad proof := [a]\ ded\ a.$
$\}.$

Fig. 10. $Mizar \to ZFC$ view

A Mizar $proof$ maps to a ZFC ded which makes proofs translatable from one signature to another.

A small part of the views is presented here as Twelf code. Note that there are other subtle differences between our ZFC and $Mizar$ which require some extra attention. For example in our ZFC_FOL the introduction and elimination rules for or and imp

```
view HiddenSem : HIDDEN → Mizar_ZFC = {
   include MizSem.
   set                := [x] true.
   in                 := [A][B] (cwhichA) in (cwhichB).
   Element_of         := [A][x] x in (cwhich A).
   Subset_of          := [A][x] x in (cwhich A).
   SUBDOMAIN_of := [A][x] x in (cwhich A) and (nonempty x).
}.
```

Fig. 11. $Hidden \rightarrow ZFC$ view

are given as axioms. However in $Mizar\ or$ and imp are defined using and and not as described in section 5.2. This causes some difficulty when doing the view. When $Mizar$'s and is mapped to FOL's and Twelf immediately understands and can translate it. This makes mapping some other definition that uses and inside considerably easier. In the case of or and imp however this is not done automatically by Twelf and it needs to be explicitly proved each time we use it. For example eqI in our $Mizar$ is defined (as in Mizar) in the following way :

$eqI : proof\ for\ [x : be\ set]\ for\ [y : be\ set]((for\ [z : be\ set] z\ in\ x\ iff\ z\ in\ y)\ implies\ (x\ eq\ y))$.

In our ZFC, eqI is defined quite similarly:

$eqI : (\{x\} ded\ x\ in\ A \rightarrow ded\ x\ in\ B) \rightarrow (\{x\} ded\ x\ in\ B \rightarrow ded\ x\ in\ A) \rightarrow ded\ A\ eq\ B$.

From this we can easily prove (in ZFC) that :

$eqI : ded\ forall\ [x]\ forall\ [y]((forall\ [z] z\ in\ x\ equiv\ z\ in\ y)\ imp\ (x\ eq\ y))$.

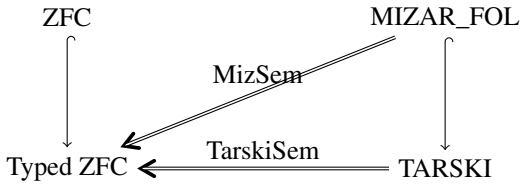


Fig. 12. Mizar → ZFC

It is also quite straightforward to move this from untyped to typed since the $Class$ for sets is the trivial class – $Class[x]true$ –. Since $proof$ maps to ded , for to $forall$ and eq to eq this looks like the equivalent of the Mizar statement. However because iff and $implies$ are different in Mizar from ZFC they do not map to $equiv$ and imp respectively. To fix this and other subtle problems we created a signature $Mizar\_ZFC$ (similarly to what we did in Isabelle) where we defined some extra theorems to make mappings easier to do and understand.

For example we defined orM in the same way as the or of Mizar and then proved that it is equivalent to our or.

$$
\begin{array}{lll}
orM & : & prop \to prop \to prop \\
 & = & [a][b]\ not\ ((not\ a)\ and\ (not\ b)). \qquad \texttt{infix right 10}\ orM. \\
orZM & : & ded\ A\ or\ B \to ded\ A\ orM\ B \\
 & = & [p]\ orE\ p\ ([q]\ notI\ ([r]\ notE\ (andEl\ r)\ q)) \\
 & & ([q]\ notI\ ([r]\ notE\ (andEr\ r)\ q)). \\
orMZ & : & ded\ A\ orM\ B \to ded\ A\ or\ B \\
 & = & [p]\ orE\ (tnd)\ ([q]\ orIr\ q)\ ([r]\ orIl \\
 & & (not\_idem(notI\ ([s]\ notE\ p\ (andI\ s\ r))))). \\
orM\_prop & : & ded\ (A\ orM\ B)\ equiv\ (A\ or\ B) \\
 & = & equivI\ orMZ\ orZM.
\end{array}
$$

We created similar equivalences for imp and $equiv$ and then used them to infer the proper mapping from Mizar to ZFC.

This means that the mapping for eqI is :

$$
\begin{array}{l}
\texttt{view}\ TarskiSem : Tarski \to Mizar\_ZFC = \{ \\
\quad eqI := ForallI\ [X]\ ForallI\ [Y]\ (impZM\ (impI\ ([p] \\
\qquad eqI([x]\ equivEl\ (equivMZ\ (forallE\ (forMZ\ p)\ x))) \\
\qquad ([x]\ equivEr\ (equivMZ\ (forallE\ (forMZ\ p)\ x)))))) \\
\}.
\end{array}
$$

Fig. 13. $Tarski \to ZFC$ view

6 Conclusion

We have formalized three foundations of mathematics in LF: ZFC, Mizar's Tarski-Grothendieck set theory and Isabelle/HOL. Moreover, we presented a way to recover type theory from untyped set theory and two translations from Isabelle/HOL and Mizar's Tarski-Grothendieck set theory to ZFC by using the recovered type theoretical semantics. The morphisms are presented in Fig. 14 where double arrows represent views.

Future work will focus on three research directions.

Firstly, we can formalize more foundations and add more translations by extending this approach to other systems like Coq [3], Scunak [5], PVS [23] or Matita [1].

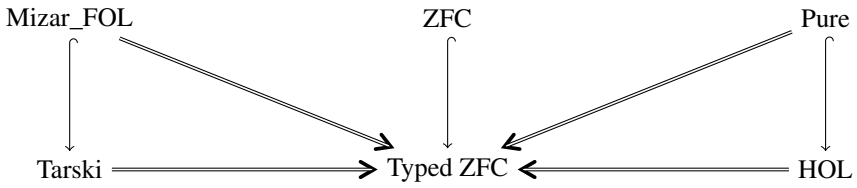


Fig. 14. Mizar → ZFC ← Isabelle/HOL

Secondly, we can use ongoing work regarding the implementation of hiding in LF [17] to perform translations with filtering (or *partial morphisms*) when a full translations is not possible because the target theory is simply not strong enough. This would allow us to do the reverse views for foundations we have presented here ($ZFC \rightarrow Mizar$ and $ZFC \rightarrow Isabelle$) and use morphism composition to infer views to and from Mizar and Isabelle which would allow us to translate between any two of these theories in any direction.

Finally, we can integrate our translations with existing and ongoing work. The morphisms described here are only withing the LF logical framework and we still require a translation from Mizar and Isabelle to LF to be able to use the system. However,Twelf is able to generate OMDoc – a markup model and data model for Open Mathematical Documents [16] – from Twelf code as well as Twelf code from OMDoc. In addition translations from Isabelle and Mizar to OMDoc are currently in progress [17].

In conclusion, together with the work described above we can help provide interoperability between existing mathematical libraries and theorem provers resulting in a unified corpus of mathematical knowledge.

References

1. A. Asperti, C. Sacerdoti Coen, E. Tassi, and S. Zacchiroli. Crafting a Proof Assistant. In T. Altenkirch and C. McBride, editors, *TYPES*, pages 18–32. Springer, 2006.
2. H. Barendregt. Lambda calculi with types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Oxford University Press, 1992.
3. Y. Bertot and P. Castéran. *Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.
4. N. Bourbaki. *Theory of Sets*. Elements of Mathematics. Springer, 1968.
5. C. Brown. Combining Type Theory and Untyped Set Theory. In U. Furbach and N. Shankar, editors, *International Joint Conference on Automated Reasoning*, pages 205–219. Springer, 2006.
6. Chad E. Brown. Scunak Users Manual, 2006.
7. A. Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5(1):56–68, 1940.

8. W. Farmer, J. Guttman, and F. Thayer. Little Theories. In D. Kapur, editor, *Conference on Automated Deduction*, pages 467–581, 1992.
9. A. Fraenkel. The notion of 'definite' and the independence of the axiom of choice. 1922.
10. Georges Gonthier. A computer-checked proof of the Four Colour Theorem. In *Technical Report, Microsoft Research, Cambridge, United Kingdom*.
11. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.
12. John Harrison. The hol light manual (1.1), 2000.
13. L. Henkin. Completeness in the Theory of Types. *Journal of Symbolic Logic*, 15(2):81–91, 1950.
14. Joe Hurd. OpenTheory: Package management for higher order logic theories. pages 31–37.
15. C. Keller and B. Werner. Importing HOL Light into Coq. In M. Kaufmann and L. Paulson, editors, *LNCS*, volume Proceedings of the Interactive Theorem Proving conference, LNCS 2010, Edinburgh, UK, July 11-14, 2010, 2010.
16. M. Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.2)*. Number 4180 in Lecture Notes in Artificial Intelligence. Springer, 2006.
17. Latin: Logic atlas and integrator. `https://trac.omdoc.org/latin/`.
18. P. Martin-Löf. An Intuitionistic Theory of Types: Predicative Part. In *Proceedings of the '73 Logic Colloquium*, pages 73–118. North-Holland, 1974.
19. S. McLaughlin. An Interpretation of Isabelle/HOL in HOL Light. In N. Shankar and U. Furbach, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*. Springer, 2006.
20. Michal Muzalewski. An Outline of PC Mizar, 1993.
21. P. Naumov, M. Stehr, and J. Meseguer. The HOL/NuPRL proof translator - a practical approach to formal interoperability. In *14th International Conference on Theorem Proving in Higher Order Logics*. Springer, 2001.
22. S. Obua and S. Skalberg. Importing HOL into Isabelle/HOL. In N. Shankar and U. Furbach, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*. Springer, 2006.
23. S. Owre, J. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, pages 748–752. Springer, 1992.
24. L. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.
25. L. Paulson and M. Coen. Zermelo-Fraenkel Set Theory, 1993. Isabelle distribution, ZF/ZF.thy.
26. F. Pfenning, C. Schürmann, M. Kohlhase, N. Shankar, and S. Owre. The Logosphere Project, 2003. `http://www.logosphere.org/`.
27. F. Rabe. Representing Isabelle in LF. 2010. to appear in proceedings of LFMTP.
28. F. Rabe and C. Schürmann. A Practical Module System for LF. In J. Cheney and A. Felty, editors, *Proceedings of the Workshop on Logical Frameworks: Meta-Theory and Practice (LFMTP)*, pages 40–48. ACM Press, 2009.

29. F. Rabe and C. Schürmann. A practical module system for LF. In *Proceedings of the Workshop on Logical Frameworks Meta-Theory and Practice (LFMTP)*, 2009.
30. B. Russell. Mathematical Logic as Based on the Theory of Types. *American Journal of Mathematics*, 30:222–262, 1908.
31. Patrick Suppes. *Axiomatic Set Theory*. Dover Publications, 1972.
32. A. Tarski. Über Unerreichbare Kardinalzahlen. *Fundamenta Mathematicae*, 30:176–183, 1938.
33. Alfred Tarski. *On the well ordered subsets of every set*. Fundamenta Matematicae, 1939.
34. A. Trybulec. Tarski Grothendieck Set Theory. *Journal of Formalized Mathematics*, Axiomatics, 1989.
35. A. Trybulec and H. Blair. Computer Assisted Reasoning with MIZAR. In A. Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 26–28, 1985.
36. J. von Neumann. Eine Axiomatisierung der Mengenlehre. *Journal für die reine und angewandte Mathematik*, 154:219–240, 1925.
37. M. Wenzel. The Isabelle/Isar Reference Manual, 2009. `http://isabelle.in.tum.de/documentation.html`, Dec 3, 2009.
38. A. Whitehead and B. Russell. *Principia Mathematica*. Cambridge University Press, 1913.
39. Freek Wiedijk. Mizar : An Impression. `http://www.cs.ru.nl/~freek/mizar/mizarintro.ps.gz`, 1999.
40. Flyspeck project. `http://code.google.com/p/flyspeck/`.
41. Isabelle Project. `http://www.cl.cam.ac.uk/research/hvg/Isabelle/`.
42. Mizar Project. `http://www.mizar.org`.
43. MML Query. `http://mmlquery.mizar.org/`.
44. Scunak project. `http://www.ags.uni-sb.de/~omega/software/scunak/index.html`.
45. Twelf sources. `https://svn.kwarc.info/repos/twelf/set_theories/` and `https://svn.kwarc.info/repos/twelf/projects/isabelle`.
46. E. Zermelo. Untersuchungen über die Grundlagen der Mengenlehre I. *Mathematische Annalen*, 65:261–281, 1908. English title: Investigations in the foundations of set theory I.