# Reconstruction of the Mizar Type System in the HOL Light System

O. Kunčar

Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic.

**Abstract.** The Mizar system is a system for formalization of mathematics. It contains a relatively sophisticated and rich type system, which makes formalization of mathematics in Mizar more intuitive than in other systems. On the other hand, the Mizar type system is very complex and together with obscure implementation of the Mizar verifier there is concern about correctness of the Mizar system. One of the possible solutions is to translate the Mizar Math Library (MML) to other systems for formalization of mathematics and reverify MML in them. The HOL Light system has been chosen and the necessary first step of proposed translation is to reconstruct the Mizar type system in the HOL Light system, which is the aim of the presented work. The reconstruction is not easy because of complexity of the Mizar type system and different types of logic used in both systems. The basic idea is to represent Mizar types as predicates and express the dynamic part of the Mizar type system as proved theorems in HOL Light. The reconstruction was partialy implemented. The paper gives necessary introduction and summarizes main principles and obstacles encountered during reconstruction.

## Introduction

The correctness of mainstream mathematical proofs is almost never established by formal means, but rather by informal discussion between mathematicians and peer review of papers. Examples of failure of this process are well-known, for example the case of Four-Color Theorem. The first proof by Kempke in 1879 was accepted for a decade before an error was found. Moreover, mathematics is becoming increasingly specialized, and some papers are read by few if any people other than their authors. Many proofs are huge — the proof of the graph minor theorem by Robertson and Seymour is about 500 pages long; some others are based extensively on computer checking of cases — for example Four-Colour Theorem or Kepler Conjecture [Harrison, 2008].

So where humans make mistakes, computers are sometimes able to prevail. Therefore many computer systems for formalization and verification of mathematics have been developed. They are based on several types of formalism and use different type systems. One of the most prominent systems is the Mizar system. As of June 2010, it contained about 9500 definitions and 50.000 verified theorems [mml] including also several well-known theorems, e.g. Alexander's Lemma, the Banach Fixed Point Theorem, Fermat's Little Theorem, the Fundamental Theorem of Algebra, the Goedel Completeness Theorem, the Jordan Curve Theorem for special polygons, and many others [Trybulec, 2006].

This paper presents author's proposal of a reconstruction of the Mizar type system to the HOL Light system. There are two issues that serve as motivation for the reconstruction:

(i) The Mizar type system is richer than the HOL Light type system — it contains structures, dependent types, subtyping, attributes, predicates and functors overloading, and many others. These features allow much better to capture a way how mathematicians use mathematical objects and how they categorize them [Markus Wenzel, 2002]. This fact is not accidental because a design of the type system, or more precisely the whole Mizar, is based on the study that describes how mathematicians write and presents their mathematical knowledge in scientific articles. Mizar made itself different from other systems for formalization of mathematics by this approach [Schwarzweller, 2007]. Therefore it is very natural to try to translate the rich and intuitive type system to other systems in order to take advantage of its valuable features.[1]

(ii) On the other hand, disadvantage of Mizar is quite complex implementation of the Mizar verifier, the source code of which is not even publicly available. Moreover, it is not even possible to write out

[1] We suppose that translation preserves or simulates the mentioned valuable features of Mizar in other system.

inferences of atomic proof steps[2], which the verifier found out during verification, in order to verify these inferences by another independent tool [Josef Urban, 2009]. These issues arouse natural concern that there can be a error in the Mizar verifier, which can lead to the situation where incorrect proofs could be verified. It turns out that one of the natural solutions is to translate the proofs written in the Mizar syntax to another system and there verify them again. Provided the translation is correct, we can lower the probability that there is an error in the Mizar verifier. Reconstruction of the Mizar type system in the HOL Light system is one of the prerequisites of realization of the above-mentioned translation. Moreover, as it is described in [John Harrison, 2006; Harrison, 2007], the HOL Light system uses the LCF methodology in implementation of its verifier, and therefore it is possible to assert correctness of this verifier with high certainty.

## The Mizar type system

Semantics of Mizar is based on set theory. It is important to note that it is untyped set theory. Despite that fact, Mizar has types. But these types are not properties of objects of the Mizar theory, but they are properties of terms of the Mizar language. So Mizar is not based on any type theory [Wiedijk, 1999]. Every variable in Mizar has to have its type. Every term has a unique type, which can be computed from a type declaration of its free variables [Bancerek, 2003]. Mizar types are composed from modes and attributes. The following description will be quite informal because there is not enough space to provide completely formal description of the complex Mizar type system.

*Modes* define dependent types[3]. Here is an example of a definition of a mode in the Mizar language:

**definition**
  **let** $a_1, a_2$ **be** set;
  **mode** Function **of** $a_1, a_2$ **is** ...;
**end**;

In this definition we define new mode Function, which has two parameters ($a_1$ and $a_2$) of type set. So Function is an dependent type, which depends on two terms of type set. Detailed definition was omitted, but is intuitively clear that it defines a function from set $a_1$ to set $a_2$. We get so called radix types by instantiation modes by terms. What follows is an example using the above-defined mode Function:

Function **of** NAT, NAT
Function **of** $A, B$

In the first example is mode Function applied to two terms NAT, which denote the set of natural numbers. The whole radix type then represents a function from natural numbers to natural numbers. The second example shows that we can use also variables as terms. As noted earlier, every variable has to have its type, therefore these variables unavoidably have been declared earlier with some type — necessarily with type set in order for our example to be correct. Type set is a built-in type of Mizar and it is the only built-in type in Mizar. Every radix type except for type set has a unique supertype. The supertype is specified in the definition of a corresponding mode, and the new mode is defined by a condition that yields a subset of the supertype. Because type set is the only built-in type and it is the only type that does not have a supertype, it is the most general type in the hierarchy of the Mizar radix types. So the hierarchy of radix types has the shape of a tree, and there is type set in the root of the tree [Wiedijk, 1999]. Here is provided a self-explanatory example concerning an idea of supertypes and generally subtyping:

$$\text{Function } \textbf{of } X, Y \preceq \text{Relation } \textbf{of } X, Y \preceq \text{set}$$

Radix types are not sufficiently strong, so it has to be introduced another kind of feature – *attributes*. The following is a motivating example for introducing attributes. Let $X$ is a variable of type TopSpace corresponding to a topological space. We would like to define a new mode Open-Closed-Subset **of** $X$, and in the definition we have to state a supertype of this new defined mode. Suppose we have already defined modes Open-Subset **of** $X$ and Closed-Subset **of** $X$. It is natural requirement that radix types

[2]These not further decomposable steps are marked in Mizar by the keyword **by**.

[3]Dependent types are types that are parametrized by terms. So they contain in their definition term variables. For example Nat[$n$] denotes natural numbers lower than $n$. An example of an instantiated dependent type is type Matrix[5, 6], which denotes matrices of size $5 \times 6$. Because variables are also terms, we can state as an example the following type List[$n$], which denotes all lists of the length $n$ where $n$ is somewhere earlier declared variable.

made from the new defined mode Open-Closed-Subset **of** $X$ have as supertype Open-Subset **of** $X$ and also Closed-Subset **of** $X$. But it is not possible because as was stated in the previous paragraph, every new defined mode can have only one supertype [Trybulec, 1993]. The solution is to define new attributes open and closed and write the following type:

$$\text{open closed Subset } \textbf{of } X$$

Then our natural requirement trivially holds:

$$\text{open closed Subset } \textbf{of } X \preceq \text{closed Subset } \textbf{of } X$$
$$\text{open closed Subset } \textbf{of } X \preceq \text{open Subset } \textbf{of } X$$

The rule is: if a type contains a set of attributes that is a subset of attributes of another type, then the first type is a supertype of the second type[4].

So far we have used a notion of the Mizar type quite intuitively. Mizar types are defined inductively: (i) Every radix type is a type. (ii) Every notation $\alpha_1\ \alpha_2 \ldots \alpha_k\ \theta$ is a type[5] where $\alpha_1, \alpha_2, \ldots, \alpha_k$ are attributes and $\theta$ is already known type [Kunčar, 2009]. There are other interesting features of the Mizar type system especially concerning the dynamic part of the type system. Mizar allows automatic inference of attributes, i.e. some set of attributes infers some other set of attributes. A user can define, for example, the following inference:

$$\text{empty set} \quad \dashrightarrow \quad \text{finite set}$$

The integral part of the definition is a proof of the correctness of the new defined rule. Another example is inference of attributes of a result type of a function. Let $X$ and $Y$ have type finite set then a user can define an inference as follows:

$$X \setminus/ Y \quad \dashrightarrow \quad \text{finite}$$

It says nothing else than that union of two finite sets is also finite.

## Reconstruction

The HOL Light system offers its own mechanism for creating new types. Suppose we have a non-empty set of values of type $\theta$ defined by its characteristic function `P` of type $\theta \to$ `bool`. Then by a proper call of function `new_type_definition` it is possible to define a new type that represents just values given by characteristic function `P`. That is almost the same mechanism like in Mizar. But the most significant difference is that Mizar uses dependent types. And a dependent type can contain a variable that is quantified in a formula that contains the dependent type. For example:

$$\textbf{ex } x \textbf{ being } \theta \textbf{ st for } y \textbf{ being } \theta'[x] \textbf{ holds } \psi(y)$$

If we substitute a witness for $x$ in existential quantification, then the type of variable $y$ changes because this type depends on variable $x$. And that causes a problem because if $\theta'[x]$ were a type of HOL Light, then the type of variable $y$ would be fixed, and it would not be changed by logical inferences, e.g. by substituting a witness for $x$ [Kunčar, 2009].

It seems that the HOL Light type system cannot be used directly. We have to simulate the Mizar type system by the logical means of HOL Light. The inspiration is Bart Jacobs [1993] where authors proposed translation of dependent types — formulated by Martin-Loef in his intuitionistic theory of types — into the HOL system. The base idea is simple — to represent types as predicates. We will use this base idea and we will try to modify it for our situation of translating the Mizar type system to HOL Light. Every function with the signature $\mathtt{P} : \tau \to \mathtt{bool}$ is considered to be a predicate in HOL Light. It claims that an argument of type $\tau$ has or does not have the property `P`. But we use quite special signature of predicates for our translation:

$$\mathtt{P} : \underbrace{\sigma \to \cdots \to \sigma}_{\text{n-times}} \to \sigma \to \mathtt{bool}$$

[4]The second condition is that radix types of the mentioned types have to be the same, or the radix type of the second type is a subtype of the radix type of the first type.

[5]Strictly speaking, every notation where attributes are applicable on the type $\theta$, but it is not important in this quite informal introduction to the Mizar type system.

Where $\sigma$ is a fixed atomic type[6] [7], which we choose for the translation, and it represents HOL Light objects that Mizar objects are translated onto. In other words, type $\sigma$ corresponds to type set of Mizar. So we consider predicates that have parameters — the first $n$ arguments of predicate `P` — and these parameters coresponds to parameters of a translated type.

The situation can be quiet unclear because we have several notions of type, which belong to different metalevels. The situation is depicted on figure 1. OCaml is a programming language, which the HOL

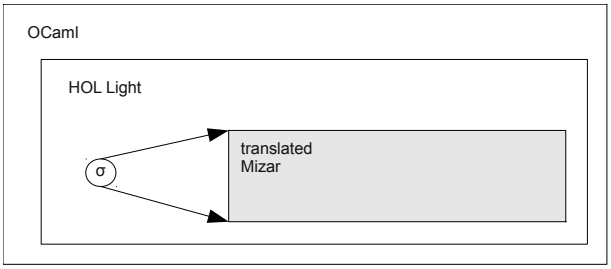


**Figure 1.** Metalevels of type systems

Light system is implemented in. It plays a role of metalanguage, which we use to reason about logic in HOL Light.[8] Ocaml as well as many other functional languages contains types. The HOL Light system contains types too. They are types of terms that are used in its logic. So they are types of expressions of the HOL Light language. The situation gets more complicated if we start thinking about types translated from Mizar. They are an additional level and the third "kind" of types. These types will be simulated by predicates of the HOL Light system, and they will be assigned to objects of the HOL Light logic that have the fixed chosen type (in the meaning the type of the HOL Light system) $\sigma$.

Let us give an example, which informally clarify the idea of translation. Suppose we have the following type qualification in Mizar:

$$y \textbf{ is } \mathsf{empty}\ \mathsf{Element} \textbf{ of } X$$

This qualification will be translated to HOL Light as follows:

$$\mathtt{I_{empty}\ y\ /\backslash\ I_{Element}\ X\ y}$$

Symbols $\mathtt{I_{empty}}$ and $\mathtt{I_{Element}}$ are predicates that correspond to attribute empty and to radix type Element respectively. Types of these predicates are as follows:

$$\mathtt{I_{empty}} : \sigma \to \mathtt{bool}$$
$$\mathtt{I_{Element}} : \sigma \to \sigma \to \mathtt{bool}$$

So $\mathtt{I_{empty}}$ is a predicate that assigns to objects of type $\sigma$ property of being empty and predicate $\mathtt{I_{Element}}$ assigns to objects of type $\sigma$ property of being a subset of `X`, where `X` is its first parameter. In the following table there is an informal summary of how the translation is done:

| Mizar | reconstruction in the HOL Light system |
|---|---|
| radix type | predicate |
| attribute | predicate |
| type | conjunction of translated attributes and a translated radix type |
| declaration assigning to variable $x$ type $\theta$ | assumption that `x` satisfies a formula corresponding to a translated type $\theta$ |

The translation was described more formally in author's master thesis Kunčar [2009] by defining formal mapping T, which maps Mizar formulas to the HOL Light language. The heart of formal description is a pair of auxiliary mappings – $\mathrm{T_{type}}$ and $\mathrm{T_{attr}}$, which map radix types and attributes respectively

[6] So $\sigma$ is neither a functional type nor a type variable.

[7] In implementation is chosen a new type `set`, which is defined by the HOL Light function `new_type`.

[8] Using OCaml functions we can add new definitions (e.g., the above-mentioned functions for introducing new types — `new_type` and `new_type_definition`), manipulate with theorems, etc.

to corresponding predicates in HOL Light. For example, suppose that $\theta$ is a radix type of the following notation $\Gamma$ **of** $t_1, \ldots, t_n$, then translation of the qualification that $t$ has type $\theta$ is defined as follows:

$$\mathrm{T}_{\mathrm{type}}(\theta, t) = \mathtt{I}_\Gamma(\,\mathrm{T}(t_1))\ldots(\,\mathrm{T}(t_n))(\,\mathrm{T}(t))$$

Where $\mathtt{I}_\Gamma$ is an identifier corresponding to the Mizar mode identifier $\Gamma$.[9] Then translation of general type qualification that $t$ has type $\theta$ where $\theta$ is the type of the following notation: $\alpha_1 \ldots \alpha_k\ \tau$, is translated as:

$$\mathrm{T}_{\mathrm{type}}(\theta, t) = \mathrm{T}_{\mathrm{attr}}(\alpha_1, \tau, t)\ \mathtt{/\backslash}\ \ldots\ \mathtt{/\backslash}\ \mathrm{T}_{\mathrm{attr}}(\alpha_k, \tau, t)\ \mathtt{/\backslash}\ \mathrm{T}_{\mathrm{type}}(\tau, t)$$

Let us show for illustration the definition (using mapping $\mathrm{T}_{\mathrm{type}}$) of T for quantified formulas:

$$\begin{aligned}
\mathrm{T}(\textbf{ex}\ x\ \textbf{being}\ \theta\ \textbf{st}\ \psi) &= \mathtt{?x}.\,\mathrm{T}_{\mathrm{type}}(\theta, \mathtt{x})\ \mathtt{/\backslash}\ \mathrm{T}(\psi)\\
\mathrm{T}(\textbf{for}\ x\ \textbf{being}\ \theta\ \textbf{holds}\ \psi) &= \mathtt{!x}.\,\mathrm{T}_{\mathrm{type}}(\theta, \mathtt{x})\ \mathtt{==>}\ \mathrm{T}(\psi)\\
\mathrm{T}(\textbf{for}\ x\ \textbf{being}\ \theta\ \textbf{st}\ \phi\ \textbf{holds}\ \psi) &= \mathtt{!x}.\,\mathrm{T}_{\mathrm{type}}(\theta, \mathtt{x})\ \mathtt{==>}\ \mathrm{T}(\phi)\ \mathtt{==>}\ \mathrm{T}(\psi)
\end{aligned}$$

So far we have been talking about static part of the Mizar type system, and how to translate it to HOL Light. But there are relations between types — some types are subtypes of other types. The described translation does not in any way capture the fact that, for example, Function **of** $X, Y$ is a subtype of Relation **of** $X, Y$. Moreover, as was sketched in the previous section, Mizar contains a mechanism of automatic inferences of attributes. So when a term has some declared type, this type has to be "rounded up" — i.e., go through all the defined and relevant inference rules for attributes, and put iteratively inferred attributes to the original type according to the rules. The general idea of how to capture the dynamic part of the Mizar type system is to use proved sentences in HOL Light – i.e, to capture relations between types by facts — proved sentences — about them. For example, the fact that $\theta_1$ is a subtype of $\theta_2$ is captured by the following sentence in HOL Light:

$$\mathtt{I}_{\theta_1}\ \mathtt{x} \vdash\ \mathtt{I}_{\theta_2}\ \mathtt{x}$$

## Implementation

Some parts of the proposed translation were implemented to verify feasibility of the translation and to catch even technical issues of it. Done work of the translation contains necessary data structures and infrastructure for adding newly defined types, functions and predicates. There are also functions that can answer non-trivial questions concerning dynamic part of the type system – e.g., "is this type subtype of another type?". Details can be found in Kunčar [2009]; here we mention only some facts about the Mizar checker and about integration of the proposed translation and its implementation into the checker.

The prover of atomic steps (in Mizar is called checker) is invoked every time the keyword **by** is hit. The aim of the authors of Mizar was to make the checker so strong that its performance would correspond to the intuitive notion of evident steps in mathematical proofs. So the checker should be able to prove steps that are evident in regular mathematical proof, and there is no need to write them out into details. On the other hand, the checker should not be too strong because its strength could motivate a user to make huge steps that would make a proof unclear. The idea of evident steps is formalized by restriction that every universally quantified formula can be instantiated only once during the checker's computation. If we want to make the checker use a formula more than once, we have to write the formula more times as an assumption of the **by** step. This approach is not so artificial because it is frequent practice to say: "and now we use transitivity twice ..." Wiedijk [2001]. The Mizar checker was implemented in the HOL Light system with some limitations by John Harrison.[10] The implementation is based on the description of the checker in Freek Wiedijk [2000], but it does not contain any part of the Mizar type system. Integration of the translated Mizar type system to the John Harrison's implementation of the Mizar checker would be important step in reimplementing the Mizar system in another system for formalization of mathematics. The work on the mentioned integration has been started by author of this paper and it is still in progress.

---

[9] Mizar identifiers have to be decorated because there are separated namespaces for modes, attributes, predicates, and functors in Mizar, but in HOL Light there is only one dimensional namespace for identifiers. Moreover, Mizar allows overloading of identifiers and also quite free lexical conventions.

[10] Except the fact that HOL Light uses the proof kernel, partial implementation of the Mizar checker was another reason why HOL Light was chosen for reconstruction of the Mizar type system.

The integration affects two places in the checker. The checker works as disprover — so the checker attempts to find inconsistency in the given set of assumptions and the negation of the given conjecture, and during this search it uses unification [Freek Wiedijk, 2000]. In unification the checker tries to instantiate variables, and this is the first place where to incorporate the type system because we could eliminate some instantiations due to type inconsistency between a type of a term and a type of a variable, which is being instantiated by the term. This is where the dynamic part of the type system comes into play because we want to know whether some type is not a subtype of another type, which would mean they are inconsistent. This cut off of inconsistent instantiations seems to speed up the checker. Further, the checker tries to compute classes of equal terms in its part where it deals with equality [Freek Wiedijk, 2000]. That is the second place where types have to be incorporated because we have to forbid equality of two terms that are type inconsistent. Speed up is here expected as well.

## Conclusion

Mizar is one of the most prominent systems for formalization of mathematics — it contains very sophisticated and complex type system, which makes formalization of mathematics more intuitive. In this paper the reconstruction of this type system into the HOL Light system was proposed. The key idea is to represent types as predicates, and the dynamic part of the type system is captured by proved sentences in HOL Light. The reconstruction was partially implemented, and integration of the implementation to the Mizar checker in HOL Light is still in progress. The outlook for the future work is to finalize the implementation and integration, and to design new type systems for the first order logic in the environment of the higher order logic using new experience gained during this work.

## References

Mml query (home page), URL `http://mmlquery.mizar.org/`.

Bancerek, G., On the structure of mizar types, *Electronic Notes in Theoretical Computer Science*, *85*, 69 – 85, mathematics, Logic and Computation (Satellite Event of ICALP 2003), 2003.

Bart Jacobs, T. M., Translating dependent type theory into higher order logic, in *Typed Lambda Calculi and Applications*, vol. 664 of *Lecture Notes in Computer Science*, pp. 209–229, Springer Berlin / Heidelberg, 1993.

Freek Wiedijk, A. T., *CHECKER - Notes on the Basic Inference Step in Mizar*, 2000.

Harrison, J., *HOL Light Tutorial*, 2007.

Harrison, J., Formal proof – theory and practice, *Notices of the American Mathematical Society*, *55*, 1395–1406, 2008.

John Harrison, Konrad Slind, R. A., HOL, in *The Seventeen Provers of the World*, vol. 3600 of *Lecture Notes in Computer Science*, pp. 11–19, Springer Berlin / Heidelberg, 2006.

Josef Urban, G. S., ATP-based Cross-Verification of Mizar Proofs: Method, Systems, and First Experiments, *Mathematics in Computer Science*, *2*, 231–251, 2009.

Kunčar, O., *Systems for formal mathematics*, Master's thesis, Charles University, Faculty of Mathematics and Physics, 2009.

Markus Wenzel, F. W., A comparison of Mizar and Isar, *J. Automated Reasoning*, *29*, 389–411, 2002.

Schwarzweller, C., Mizar Attributes: A Technique to Encode Mathematical Knowledge into Type Systems, *Studies in Logic, Grammar and Rhetoric*, *10(23)*, 387–400, 2007.

Trybulec, A., *Some Features of the Mizar Language*, 1993.

Trybulec, A., Mizar, in *The Seventeen Provers of the World*, vol. 3600 of *Lecture Notes in Computer Science*, pp. 20–23, Springer Berlin / Heidelberg, 2006.

Wiedijk, F., *Mizar: An Impression*, 1999.

Wiedijk, F., Mizar light for HOL light, in *Theorem Proving in Higher Order Logics*, vol. 2152 of *Lecture Notes in Computer Science*, pp. 378–393, Springer Berlin / Heidelberg, 2001.