# Translating Mathematical Vernacular into Knowledge Repositories

Adam Grabowski[1] and Christoph Schwarzweller[2]

[1] Institute of Mathematics, University of Białystok,
ul. Akademicka 2, 15-267 Białystok, Poland
adam@math.uwb.edu.pl
[2] Department of Computer Science, University of Gdańsk,
ul. Wita Stwosza 57, 80-952 Gdańsk, Poland
schwarzw@math.univ.gda.pl

**Abstract.** Defining functions is a major topic when building mathematical repositories. Though relatively easy in mathematical vernacular, function definitions rise a number of questions and problems in fully formal languages (see [4]). This becomes even more important for repositories in which properties of the defined functions are not only stated, but also proved correct. In this paper we investigate function definitions in the Mizar system. Though most of them are straightforward and follow the intuition, we also found a number of examples differing from mathematical vernacular or where different solutions seem equally reasonable. Sometimes there even do not seem to exist solutions not somehow "ignoring mathematical vernacular". So the question is: Should we seek for some kind of standard, that is a "formal mathematical vernacular", or should we accept that different authors prefer different styles?

## 1 Introduction

Mathematical knowledge management aims at providing both tools and infrastructure supporting the organization, development, and teaching of mathematics on computers. Large repositories of mathematical knowledge are of major concern since they provide the user with a knowledge base of verified mathematical facts. However, this knowledge is often not easy to access due to the formal language in which it is presented and verified. On the other hand the acceptance of repositories and hence of mathematical knowledge management systems heavily relies on the way mathematics is presented to the user; thus the closer to "everyday" mathematics the used language is, the more likely users of the system will be found.

The language actually used by mathematicians, however, is rather vague and imprecise: working mathematicians use what is called the "mathematical vernacular" [3, 9], a language rather to communicate than to be completely formal. As stated by Davenport [4] "It turns out to be remarkably hard to write 'correct' mathematics in the mathematical vernacular." The reason is that the knowledge implicitly used in the vernacular must be made explicit for 'correct' mathematics. The same holds for knowledge repositories, especially if such a repository

M. Kohlhase (Ed.): MKM 2005, LNAI 3863, pp. 49–64, 2006.

is connected with a theorem prover or checker and is not just a collection of (textual) definitions and theorems. Here, for example, we do not have "obvious" special cases that "need not to be taken into accounct".

On the other hand existing theorem provers and checkers provide languages successfully used to formalize and prove numerous advanced theorems. The languages to do so, however, are usually highly specialized and hard to understand from the viewpoint of working mathematicians. The reason is that here the language has to be not only formal but also semantically exact in order to produce reliable proofs of theorems. As a consequence, there is a clash between what mathematicians and computers – that is computer scientists who design and implement theorem provers and checkers – consider comfortable. For theorem proving it might be reasonable to use languages "bizar" to a mathematician, as the goal is "simply" to find a (representation and) a computer proof for a specially chosen theorem.

In mathematical repositories the situation is somewhat different: here we look for general methods describing (and proving) knowledge from different – if not all – areas of mathematics. In addition this knowledge is to be accessed and used by non-specialists also, so that the knowledge should not be hidden by the formal language of the system. Nevertheless the language used has to semantically exact to produce reliable results. So the question is: Should we develop mathematical knowledge management systems as closely as possible to the vernacular of working mathematicians in order to please them as potential users? Or should we include other language elements or slightly different definitions in case they are more convenient from the theorem proving point of view?

In this paper we discuss this question by inspecting the Mizar language and the Mizar Mathematical Library. We focus on definitions, in particular function definitions, which are often given partially or by case distinctions (see [4]). This "impreciseness" is not further considered by mathematicians: theorems are stated without really worrying about the "easy special cases". In mathematical repositories, however, this is not possible and therefore Mizar provides language constructs to cope with such situations. However, as we will see, these do not allow for a one-to-one translation of the "mathematical vernacular", some decisions remain up to the author. In addition we also present example situations which a) do not strictly follow the "mathematical vernacular" and b) provide more elegant proving and reuse in a repository.

The plan of the paper is as follows. After a brief review how functions can be defined in Mizar in the next section, we start with an investigation of the empty set and its elements in section 3. This easy example already indicates, that there exist different possibilities to realize mathematical vernacular in repositories. That this is no accident is shown in section 4 and 5 where a number of examples from different areas such as trigonometric functions and arithmetics are presented. Problems concerning more involved topics such as modularity of repositories and ambiguities are discussed in section 6. These observations imply that maintaining and revising of repositories will stay an important topic in the

future. Section 7 discusses software built for the Mizar Mathematical Library to support this task.

## 2 Defining Functions in Mizar

Mathematical functions often cannot be defined uniformly on their domains; there are defined by case distinctions such as for example the signum function or even partially by giving additional conditions for the arguments as in the case of inverse trigonometric functions. Of course one can introduce new domains on which such functions are then totally defined; this, however, seems to be rather artificial and in addition would lead to an inflation of domains not acceptable in a mathematical repository.

The Mizar system basically provides two language constructs to cope with such situations: the `assume`-clause to express restrictions of arguments and the `if`-clause for defining case distinctions. In this section we give some introductory examples for using (and abusing) these constructs before we discuss their implications for mathematical knowledge repositories.

A standard example for restricting domains is the square root functions which is defined for non-negative real numbers only. The straightforward Mizar definition is as follows.

```
definition let a be real number;
  assume 0 <= a;
  func sqrt a -> real number means
    0 <= it & it^2 = a;
end;
```

Note, however, that this definition implies that for each application of `sqrt` a non-negative argument `a` is necessary, that is one has to show or state as an assumption that `0 <= a`. Things become more puzzled when considering for example trigonometric functions: `tan a` is defined only if `cos a` is not zero, we thus get

```
definition let a be real number;
  assume not ex k being Integer st a = Pi / 2 + k * Pi;
  func tan a -> real number equals
    sin(a) / cos(a);
end;
```

and, given `a`, to get the value `tan a` the assumption is evident and has to be shown explicitly. The situation looks different when it comes to case distinctions using the `if`-clause. Though defining functions this way requires proving consistency – the cases need not be distinct, so one has to show that the corresponding values do not contradict each other – most examples are straightforward and intuitive such as

```
definition let x,y be real number;
  func min(x,y) -> real number equals
    x if x <= y otherwise y;
end;
```

Proving theorems involving such functions is rather straightforward and fits to intuition. A prominent exception, though, is the inverse z" of a complex number z, which is usually considered as a partial function, 0" being undefined. In Mizar, however, we find that " is defined as a total function with 0" being equal to 0.

```
definition let z be complex number;
  func z" -> complex number means :: XCMPLX_0:def 7
    z * it = 1 if z <> 0 otherwise it = 0;
end;
```

The point is that defining " as a partial function using assume z <> 0 would require to prove this each time the definition is used; so in order to avoid this the author decided to base the development on this "slightly different definition". Note that in Mizar division z/y of complex numbers is defined as z*y". This means that / is a total function too, and in particular that one can prove z/0 = 0 for every complex number z (including z = 0). We will see in section 4 some more implications of this definition.

Of course it is easy to "abuse" these language constructs by introducing unnecessary assumptions, the probably most prominent example is using non-empty sets where this is not necessary. So the question is not only how to provide assumptions that can be reasonably used later, but also how to avoid unneccessary assumptions in a repository.

## 3 How Many Elements Has the Empty Subset?

To start the discussion we present in this section some issues of the empty set and its elements. Though rather trivial at first sight, this illustrates well the problems arising when moving from "imprecise" descriptions to "complete formal" ones. We will see that though the definition is almost trivial, using it in the environment of a mathematical repository – that is combining the definition with other notations from set theory – needs some care.

The empty set is the set which contains no elements. Thus it is straightforward to define something like

```
definition
  func {} -> set means :: XBOOLE_0:def 1
    not ex x being set st x in it;
end;
```

Though not exactly in the scope of defining functions, we like to mention the following problem here: In mathematical repositories definitions do not stand alone; they have to be considered in the context of other notations, here for

example finite and infinite sets. Obviously, the empty set is finite. But in a repository that is not true in advance, it's just obvious in the "mathematical vernacular". Thus in principle it is possible to have objects such as

```
let X be infinite empty set;
```

Generally speaking that's no harm, because such a phrase is meaningless: it includes a contradiction, hence everything stated (and proved) for such objects is of no use. On the other hand the acceptance of a repository in which this is possible is at least questionable. Thus such "contradictable" objects should be ruled out. Therefore Mizar does not allow empty types: before using an object of type `infinite empty set` its existence has to be shown in an existential cluster registration.

Now let's have a look at the elements of the empty set. In Mizar we find the definition of the type `Element of X`, where `X` is a set. It is "clear" that `x` is an `Element of X`, if $x \in X$. There are no problems if `X` is non-empty: There exists an element `x` in `X`, so the type `Element of X` is non-empty. If `X` is empty, however, there is no $x \in X$. Of course one can define the type `Element of X` for non-empty sets `X` only, ruling out the type `Element of {}`. But then each time the type `Element of` is used, one has to show that its argument is non-empty. Therefore in Mizar the type `Element of {}` is defined to be the empty set:

```
definition let X be set;
  mode Element of X means :: SUBSET_1:def 2
    it in X if X is non empty otherwise it is empty;
end;
```

This, however, does not fit to mathematical vernacular, because the empty set is not an element of the empty set; but has the advantage that the type `Element of X` is well-defined for arbitrary sets `X`, hence usable without any assumptions. We mention that though the empty set `{}` is of type `Element of {}`, this does not imply that $\{\} \in \{\}$ is provable in Mizar, that is `{}` is still the empty set. Furthermore, the Mizar checker itself infers that $x \in X$ holds if `x` is an `Element of X` and `X` is non-empty. So we see that even a notion as "obvious" as the empty set calls for basic decisions – especially concerning types and their implications for later proving – when being formalized, that is when moving from mathematical vernacular into a mathematical repository.

## 4 Special Functions

In this section we consider mainly the definition of trigonometric functions in Mizar. Interestingly, we can find different approaches, one following the intuition and another one using that the inverse of `0` is `0`. First, the logarithm of real numbers `a` and `b` is defined using exponentiation, in Mizar defined as a functor `to_power` (see section 5.3). Here, the usual "problematic" values for `a` and `b` have been ruled out using an assumption:

```
definition let a,b be real number;
  assume that a > 0 & a <> 1 and b > 0;
  func log(a,b) -> real number means :: POWER:def 3
    a to_power it = b;
end;
```

Consequently, theorems have to take these values into account, because the equality `a to_power log(a,b) = b` is valid only if the assumptions about `a` and `b` are fulfilled. We thus find theorems of the following kind.

```
theorem :: POWER:61
   a>0 & a<>1 & b>0 & c>0 implies log(a,b) + log(a,c) = log(a,b*c);
```

This approach follows what Davenport called the "conditional equation approach" in [4]. The advantage is that it is close to textbook mathematics (though assumptions in a book are often not stated thoroughly) and makes the necessary assumptions explicit. On the other hand long lists of assumptions both decrease readability of theorems and require of course re-stating them when using such theorems in other proofs.

What can improve things a bit here is the technique using "default values" as presented in the definition of the inverse function `"`. Remembering that `/` is a total function (compare section 2), the tangent function for real numbers can be defined simply as follows.

```
definition let th be real number;
  func tan(th) -> real number :: SIN_COS4:def 1
    sin(th) / cos(th);
end;
```

which actually means that `tan(Pi/2)` is defined to be `0`. Note that given `th` we can now get the value `tan(th)` without proving `th <> Pi/2`. This also implies that a number of theorems can be stated using no assumptions, so for example

```
theorem :: SIN_COS4:2
  tan(-th) = - tan(th);
```

This may seem irritating at first sight for a reader not familiar with the basic definitions of the repository; but has the advantage that this theorem can be used without further prerequisites to be shown. Of course not all theorems can be stated this way, because `z * z" = 1` holds only if `z <> 0`. Here Mizar formalizations fall back to the conditional approach, so for example we find

```
theorem :: SIN_COS4:8
  cos(th) <> 0 implies sin(th) = cos(th) * tan(th);
```

## 5 Arithmetics and Related Issues

### 5.1 The Greatest Common Divisor

As the greatest common divisor $\mathrm{GCD}(a, b)$ is the largest number dividing both integers $a$ and $b$, according to our intuitions such number does not exist in case

of $a = b = 0$. Indeed, a quick tour through mathematical services available via WWW confirms these convictions: Wolfram's MathWorld's[1] definition of `gcd` takes only positive integer numbers as arguments; according to Wikipedia[2] both should not be zero simultaneously, similarly is the PlanetMath's[3] opinion, but we can read in Wikipedia that "it is useful to define $\gcd(0, 0) = 0$".

In the Mizar library, there are two definitions of the greatest common divisor: `hcf` for natural numbers and `gcd` with integer arguments which uses the notion of `hcf` and the absolute value in its definiens.

```
definition let k, n be Nat;
  func k hcf n -> Nat means :: NAT_1:def 5
    it divides k & it divides n &
      for m st m divides k & m divides n holds m divides it;
end;
```

Based on the above, we can easily prove that

$$a = 0 \wedge b = 0 \iff a \text{ hcf } b = 0$$

for all natural $a$ and $b$, and similarly for integers. Furthermore, claiming such definition we keep the connectedness with commutative rings, we also obtain a lattice of naturals with `gcd` and `lcm` as binary operations to be both distributive and complete.

### 5.2 The Integer Division

When inspecting the integer division in the Pascal programming language, the FreePascal compiler returns 'division by zero' error both with `div` and with `mod`. Since the Mizar system itself is coded in Pascal (and as one can easily see, some Mizar language constructions have been influenced by this programming language), we could expect a similar behaviour for the functions `div` and `mod` in the MML.

Since both are defined usually (see Wolfram's MathWorld) as:

$$m \text{ div } n = \lfloor m/n \rfloor, \quad m \bmod n = m - n\lfloor m/n \rfloor, \tag{1}$$

both share the restriction of $n \neq 0$ as usual in the literature.

This is not violation of intuitions, but the MML contains the following definitions, somewhat closer to Euclid's *Elements*:

```
definition let k, l be natural number;
  func k div l -> Nat means :: NAT_1:def 1
    ( ex t being Nat st k = l * it + t & t < l ) or it = 0 & l = 0;

  func k mod l -> Nat means :: NAT_1:def 2
    ( ex t being Nat st k = l * t + it & it < l ) or it = 0 & l = 0;
end;
```

[1] http://mathworld.wolfram.com
[2] http://www.wikipedia.org
[3] http://planethmath.org

The above definition is a variant of which we wrote earlier (something like the `if`-clause), but with a slightly different (but equivalent classically) formulation.[4] There is an agreement in the MML that 0 is an element of $\mathbf{N}$ (to have both functions natural-valued), but there isn't any within mathematics in general: MathWorld writes that "Unfortunately, 0 is sometimes also included to the list of 'natural' numbers" (as Bourbaki and Halmos do), quoting Ribenboim's as the opposition ("... whenever convenient, it may be assumed that $0 \in \mathbf{N}$").

In [8] they state that extending `mod` to omit the assumption of the division by zero is important, but they do not explain explicitly which one should be taken: the divided number or 0. Inspecting the book we discovered that if we accept the equations (1) as the new definitions of the integer division functions in the MML, we obtain $x = x \bmod 0$, and this is also claimed in [8] in many more places than the alternative $0 = x \bmod 0$.

There are contexts in which division by zero can be considered well-justified. For example, in the extended complex plane $\mathbf{C}^*$ it is defined to be a quantity known as complex infinity. This definition expresses the fact that, for $z \neq 0$, $\lim_{w \to 0} z/w = \infty$ (i.e., complex infinity). However, even though the formal statement $1/0 = \infty$ is permitted in $\mathbf{C}^*$, this does not mean that $1 = 0 \cdot \infty$, so zero does not have a multiplicative inverse. On the other hand, although $\mathbf{R} \subseteq \mathbf{C}$, it is not clear which way to go with the extensions (since to the extended set of real numbers both $+\infty$ and $-\infty$ are added and this is the case of the MML).

As a good example of the other way of definition extending we can quote `min*` as an opposition to an ordinary `min` function.

```
definition let A be finite non empty real-membered set;
  redefine func min A means  :: SFMASTR3:def 1
  it in A & for k being real number st k in A holds it <= k;
end;

definition let A be set;
  func min* A -> Nat means  :: HENMODEL:def 1
    (it in A & for k st k in A holds it <= k) if
       A is non empty Subset of NAT
          otherwise it = 0;
end;
```

These two objects are defined completely independently, but the latter became apparent to be useful when proving the Gödel's Completeness Theorem in Mizar. Theoretically, generalizing `min*` we can replace an original `min` to simplify the library a bit. Generalizing can be also interesting from a purely scientific point of view (as e.g., formalizing rough sets with tolerances as described in [7] or [6] instead of equivalence relations). But usually the loci of a definition cannot be just generalized because the information contained in it may be necessary to give the proper meaning of an introduced object.

[4] The difference between `natural number` and `Nat` (with the latter expanding to `Element of NAT`) which has origins in various treatment of element of the empty set has to be recalled here. All `Nat`s obtain the attribute `natural` automatically due to the conditional cluster mechanism.

### 5.3 The Power Operator

The consequence of introducing in parallel of similar notions (motivated by the need of having their definitions close to the literature) can be observed in the case of the definition of the power function, which is composed with the help of various power operators defined earlier in MML (`#R` is defined as the limit of sequence of rational powers of a given real number – with the assumption of the positive base, `#Z` is a integer power, with arbitrary real base).

```
definition let a, b be real number;
  func a to_power b -> real number means :: POWER:def 2
    it = a #R b if a > 0,
    it = 0 if a = 0 & b > 0,
    it = 1 if a = 0 & b = 0,
    ex k st k = b & it = a #Z k if a < 0 & b is Integer;
end;
```

Any efforts to change this definition should be made carefully, because the article with this definition is referenced in 46 other MML items 1407 times. Similar data for the other power operators: 415 references in 39 articles.

Note that this definition is an example of a definition of a partial function (and keyword `otherwise` is not used there), e.g. according to this definition we still don't know which is the value of $(-1)^{-\frac{1}{2}}$, but it gets the type `real number`.

### 5.4 Polynomials

Consider polynomials as a last example. The head term (HT) – and hence the head coefficient (HC) – of a polynomial are usually defined for non zero polynomials only (see for example [2]. From a theorem proving point of view, however, it seems convenient to define a "head term" for the zero polynomial also as follows: The head term of the zero polynomial equals the smallest term with respect to the given order. This is can be seen as an extension of the head term functor found in the literature.

```
definition
  let n be Ordinal, T be connected TermOrder of n,
      L be non empty ZeroStr, p be Polynomial of n,L;
  func HT(p,T) -> Element of Bags n means :: TERMORD:def 6
  (Support p = {} & it = EmptyBag n) or
  (it in Support p &
   for b being bag of n st b in Support p holds b <= it,T);
end;
```

This allows us to formulate theorems about head terms for arbitrary polynomials. As a consequence, when later reusing such theorems the user need not always bother that the actual polynomial is not equal 0 – just like mathematicians. For example, we get

```
theorem :: TERMORD:22
  for n being Ordinal, T being connected TermOrder of n,
      L being non trivial ZeroStr, p being Polynomial of n,L holds
    term(HM(p,T)) = HT(p,T) & coefficient(HM(p,T)) = HC(p,T);
```

if also `HC(p,T)` is defined appropriately (as head monomial `HM(p,T)` is), e.g. equals the zero element of the underlying coefficient domain.

## 6 Modularity and Ambiguity

### 6.1 Modularity of the Library

Although the fundamentals of set theory in Mizar are established in rather unflexible way (some of them are built into the verifier, e.g. the Axiom of Choice can be proved – and it is in the Mizar article [1]), the user can also modify his/her (e.g. set-theoretical) framework at very low axiomatic level. At first glance it is not strictly connected with function definitions, but certain preferences can substantially change the need of conditional definitions. As a perfect example in arithmetics of alephs we can cite the Generalized Continuum Hypothesis introduced by Josef Urban in [15].

```
definition
  pred GCH means  :: CARD_FIL:def 12
    for N being Aleph holds nextcard N = exp(2,N);
end;

theorem :: CARD_FIL:31
  GCH implies ( M is inaccessible implies M is strongly_inaccessible );
```

where $M$ is again of the type `Aleph`.

This trick may be used, e.g. to state the Brouwer Fix Point Theorem for disks on the real euclidean plane as an assumption to prove the famous Jordan Curve Theorem[5]. As the bright side of this approach to the development of the library we can point out the possibility of development of the authors' favorite parts of mathematics in which they are experts, instead of spending most time on bridging the gap between the current and the desired state of the formalization of the theory. This could attract more mathematicians and as we believe it is one of the vital aims of math-assistants and also of the MKM project. Also the research frontier could be so reached faster – which could make the machine codification of recent mathematics more egalitarian.

The modular maintenance of systems could be a solution for someone's wishes to have some meta-assumptions, but the care is advised (e.g., the Axiom of Determinacy contradicts the Axiom of Choice which is proven in the MML, so the earlier should not be accepted as such an assumption). Probably something

[5] Actually it is meaningless since Korniłowicz and Shidama proved this version of the Brouwer Fix Point Theorem in February 2005 as the `BROUWER` article accepted to the MML. The one-dimensional case is pretty old.

like the `requirements` directive with more human-friendly access and giving possibility of defining author's own modules of this type could be an attractive solution.[6]

Clearly, this can also have some impact on the knowledge exchange between different systems, according to the Sacerdoti Coen's advice in [14]: "Make implicit information explicit". Note however that the logical system standing behind the Mizar system is fixed, and Mizar developers rather do not anticipate change of this policy (e.g. from the classical into the constructive logic) in the future. Another drawback is that stating some significant or influential theorems without proofs and using them later as a starting point for further computer-checked reasoning we allow for a gray area of practically machine-unverified mathematics. This is hardly acceptable if we aim at building a knowledge repository as a block, not as a loose collection of solved problems.

Having this idea in mind one might understand the encoding of the solution of the Robbins problem just as proving set of equations given by Huntington can be derived from those of Robbins. Similarly, the problem of Sheffer-stroke-based short single axiomatization of Boolean algebras can be seen as such, involving only "|" operator and showing the equivalence with the 3-axiom system given by Sheffer.

In informal mathematics it is natural to explain that both approaches for Boolean algebras, this using disjunction and negation, and that with the Sheffer stroke are equivalent with the classical one, in terms of two binary operators and a unary complement. Authors can have different ideas for the same concept just as various books on the same topic do. But in the Mizar Mathematical Library, and – as we imagine – in an arbitrary large formal repository of mathematical knowledge it requires some work to provide a proper justification for this equivalence.

### 6.2 Ambiguities

In a distributed knowledge repository it is hard to establish a high unification level (compare $1/0 = 0$, which is provable in HOL/Mizar, its negation is provable in IMPS, both the formula and its negation are not provable in Coq, or being not a correct formula in PVS), so there is a need to exchange information about the mathlore (as in QED Manifesto they wanted to call "knowledge that is neither taught in classes nor published in monographs") which is accepted (or rather where it was rejected). As a mathlore we understand here not only basic facts which are commonly accepted, but also the formulation of definitions of basic notions.

But what to do with freshmen which are not acquainted well with the mathlore? Anyone remembers from the school that the division by zero is not allowed as a rule, and no one complained, so why think about the motivation to have

[6] As of version 7.0.04 of the Mizar system, there are five modules of this type available: `BOOLE`, `SUBSET`, `HIDDEN`, `ARITHM`, `NUMERALS`, `REAL`, where first two introduce automatization of boolean operations on sets, the latter three – calculations on numbers. The detailed exposition of the topic is included in [10].

some value for 0 divided by 0? The answer given in the MML is not obligatory in didactics: the recent policy of computer-aided instruction with Mizar is not to use the whole repository (MML), but to prepare small working environments built from scratch as described in [11], in which decisions do not depend on the Library Committee taking care of the MML. The reason is also that in this way it is independent from the Mizar library which evolves rapidly so the update during a semester could be hardly acceptable.

One of the conservative choices is to keep different definitions (and theorems, consequently) of the same notion in parallel, not to favour any distinct approach. Even then we can measure how often each of them is taken and – based on this quantitative measures – let researchers develop only the one which is used most often (as it can be done e.g. in case of `min` and `min*`) via consequently replacing other undesirable occurrences.

## 7 Improving the Library

As mathematics assistance systems are designed as a tool offering machine help for human researchers, many of the decisions about chosen approach are taken on the user's side depending on the various (subjective as a rule) criteria: elegancy, faithfulness to mathematical standards, feasibility, etc.

However, especially if the cooperation between various systems is taken into account, much improvement of a repository can be done in a highly automatic way. The quality is to be measured by statistical, so quantitative means. As it is clear however, "short" does not mean "readable" and this is a serious drawback when thinking about reusability of proofs and their clarity for people. The de Bruijn factor, which is defined by Wiedijk as the quotient of a size of formal representation by its informal original can be a dead end sometimes.

All Mizar distributions contain the bunch of programs aiming at reviewing a Mizar article and which hence may lead to the enhancing of a human work done by hand. The Library Committee of the Association of Mizar Users uses a collection of editing versions of the mentioned programs.

The software inspects a Mizar text focusing on three main activities:

- shortening and clarifying proofs;
- improving definitions' and theorems' level of generality;
- marking block and items which are just not used anymore.

Since the Mizar Mathematical Library contains knowledge which is not only declared but its correctness is also proved, there is a need for controlling of the necessity of some parts of the proofs written by human. Conditional definitions can be introduced to reflect closely the sense of the original (as the aforementioned division by zero), in many cases additional assumptions may be consequence of too weak formulation of the theorems used in the proof of its correctness, sometimes unnecessary clauses are just left accidentally (and so, polishing proofs can detect them). In all above cases, enhancing proofs can affect also the formulation of definitions via bottom-up stepwise refinement. There are few stages on which

such control (hence improvement) can be performed (and this is the case of the Mizar library):

**Irrelevant premises**
This is the most unproblematic and the most popular control which can be performed when writing a Mizar article. `relprem` reviews which references are not needed for the justification of a sentence.

**Checking unused labels**
Very often removed unused premises are just library references (for definitions and theorems proven in MML already), but sometimes the calling by a local fact is written accidentally. If any other sentence also does not use this labelled item, after the `chklab` pass such label is marked as unnecessary. Still though, the sentence can be needed in a proof via simple linking by the next one (the reserved word '`then`' in such a case).

**Inaccessible part of proofs**
The program `inacc` points out sentences which are neither labelled nor linked (elements of a proof skeleton are not marked as erroneous).

**Finding trivial proofs**
Although Mizar proofs are hierarchical (in the sense considered e.g., by Lamport), sometimes after the aforementioned transformations nested proofs can be simplified by the program `trivdemo` to a simple justification, that is to a list of references preceded by the keyword '`by`'.

**Irrelevant suppositions**
As unnecessary assumptions (in the sense of elements of proof skeleton, not just as premises) are not marked by any of the programs mentioned before as vital element of proofs, this software operates on the stage of theorem formulation than proof transformation.

This program (`relsup`) is not freely available in the distribution. Explicit formulation of some assumptions in a proof may be forced by the so-called definitional expansions and hence not used directly. They are needed however and their automatic removal could result in an error in the proof skeleton and marking them as erroneous can be highly confusing, especially for an unexperienced author.

The above ordering of these programs reflects their preferred calling sequence.

The only controversial exception of the reviewing software is `relinfer` program (so it was excluded from our enumeration), which points out the unnecessary steps in a proof (and the references should be added to the next step). It can exceptionally shorten proofs but it may result in poor readability of the text:

- some sentences which are important for the proof technically are marked as irrelevant steps, but their removal may force the user to repetition of the same library reference;
- the removal may be accidental in some sense, that is steps which are crucial for human understanding of the idea of a proof, but are still unnecessary for machine (e.g., unwinding definitions – definitional expansions). Here the tendencies to reduce the de Bruijn factor can be misleading.

We also have software which detects unused variable occurrences, irrelevant private predicates and functions, marks unnecessary type changing statements, etc.

Besides the aforementioned proof transformations which are performed very often, some other checkouts are done occasionally. There is a software which checks if there are equal theorems in the library, and what's more interesting, if a theorem is a consequence of another (although due to the large library, both use a lot of resources). The latter one is often not very unlikely: to formulate statements as equivalences is the usual mathematical practice, very often though some assumptions are needed only for one of the implications.

We still do not have any automatic control if the definitions are repeated (authors would have like to introduce independently e.g. closure operators using different structures), so we can speak about the detection of 'equal theorems' rather than 'equivalent' ones. So the role of careful peer-reviewing of a repository is very important, especially if we take into account a large repository of mathematics, written by many authors, so rather not much unified in style. Quantitative parameters of the MML (some 40 thousand of theorems and lemmas, nearly 8 thousand definitions authored by more than 170 authors) justify the necessity of continuously revising of Mizar articles.

## 8 Conclusions

In this paper we have considered how the mathematical vernacular can be realized in mathematical repositories, thereby focusing on function definitions. The inspection of the Mizar Mathematical Library has shown that its authors used a number of different styles such as the "conditional" style using partial functions or the "extension" style as used in the definition of ". Sometimes even more than one definition is available. It seems to us that these different styles exist due to a clash between (a) working in a formal language close to the mathematical vernacular which (b) is also used to prove the theorems stated. Strictly following the mathematical vernacular sometimes leads to rather tedious formal proofs, so that some authors decide to modify their definitions in order to ease the proving task.

The problem becomes more evident in a repository with a large number of developers and users: here, of course, it is impossible to have an open system without ending up with different realizations of the mathematical vernacular. Hence, should we seek for a kind of standard, that is a "formal mathematical vernacular"? Though we believe that this can be done in general, it seems hardly possible to fix all the details theorem proving introduces into our repositories. Allowing for different realizations, on the other hand, could of course decrease the acceptance of users by confusing them. Also, extending or reusing developments by other authors gets more complicated in case the vernacular of the new author does not fit to the first author's one.

What we can try to do is organize our repositories in such a way that both authors and potential users have the possibility to identify the basic decisions

theories and developments rely on. As we have illustrated this also includes the definition of functions. This is not a trivial task, because as already mentioned the large number of Mizar authors has even led to duplication of definitions or theorems. Consequently, we always have to keep track of the development by permanently revising and cleaning up our repositories. The goal must be to automate this as far as possible. A step into this direction are the Mizar tools presented in section 7. They are, however, in most cases still working on the proof transformation level, so that their further development into a "more intelligent direction" is desirable. As we understand the mathematical vernacular not only as a syntactic language, but also as the way how to shape the real mathematics (the stress on the formalized content, not only on the form – see [16]), we find it hard to establish strict guidelines (and so the question stated in the abstract remains open, although there are known direct formalizations of the traditional approach to undefinedness in the literature, e.g. [5]) for a mathematical vernacular to be feasible. We should, however, always keep in mind – especially if we try to develop systems for working mathematicians – that if we break rules accepted widely by mathematicians, this has to be sufficiently justified.

## References


1. G. Bancerek, *Zermelo Theorem and Axiom of Choice*, Formalized Mathematics, 1(2), pp. 265–267, 1990.
2. T. Becker and V. Weispfenning, *Gröbner Bases – A Computational Approach to Commutative Algebra*; Springer Verlag, 1993.
3. N.G. de Bruijn, *The Mathematical Vernacular, a language for mathematics with typed sets,* in P. Dybjer et al. (eds.), Proc. of the Workshop on Programming Languages, Marstrand, Sweden, 1987.
4. J.H. Davenport, *MKM from book to computer: a case study,* in: A. Asperti, B. Buchberger, and J. Davenport (eds.), Proc. of MKM 2003, Lecture Notes in Computer Science 2594, Springer, pp. 17–29, 2003.
5. W.M. Farmer, *Formalizing undefinedness arising in calculus,* in: D.A. Basin and M. Rusinowitch (eds.): Proc. of IJCAR 2004, Cork, Ireland, Lecture Notes in Computer Science 3097, Springer, pp. 475–489, 2004.
6. A. Grabowski, *On the computer-assisted reasoning about rough sets*, in: B. Dunin-Kęplicz et al. (eds.), Monitoring, Security, and Rescue Techniques in Multiagent Systems, Advances in Soft Computing, Springer, pp. 215–226, 2005.
7. A. Grabowski and Ch. Schwarzweller, *Rough Concept Analysis – theory development in the Mizar system,* in: A. Asperti, G. Bancerek, and A. Trybulec (eds.), Proc. of MKM 2004, Lecture Notes in Computer Science 3119, Springer, pp. 130–144, 2004.
8. R.E. Graham, D.E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison-Wesley, 1994.
9. F. Kamareddine and R. Nederpelt, *A refinement of de Bruijn's formal language of mathematics,* Journal of Logic, Language and Information, 13(3), pp. 287–340, 2004.
10. A. Naumowicz and Cz. Byliński, *Improving Mizar texts with properties and requirements*, in: A. Asperti, G. Bancerek, and A. Trybulec (eds.), Proc. of MKM 2004, Lecture Notes in Computer Science 3119, Springer, pp. 190–301, 2004.

11. K. Retel and A. Zalewska, *Mizar as a tool for teaching mathematics,* in Proc. of Mizar 30 workshop, Białowieża, Poland, 2004, available at `http://www.macs.hw.ac.uk/~retel/papers/KRetelAZalewska.pdf`.
12. P. Rudnicki and A. Trybulec, *Mathematical Knowledge Management in Mizar*; in: B. Buchberger, O. Caprotti (eds.), Proc. of MKM 2001, Linz, Austria, 2001.
13. P. Rudnicki and A. Trybulec, *On the integrity of a repository of formalized mathematics*; in: A. Asperti, B. Buchberger, and J. Davenport (eds.), Proc. of MKM 2003, Lecture Notes in Computer Science 2594, Springer, pp. 162–174, 2003.
14. C. Sacerdoti Coen, *From proof-asistants to distributed knowledge repositories: tips and pitfalls,* in: A. Asperti, B. Buchberger, and J. Davenport (eds.), Proc. of MKM 2003, Lecture Notes in Computer Science 2594, Springer, pp. 30–44, 2003.
15. J. Urban, *Basic facts about inaccessible and measurable cardinals*, Formalized Mathematics, 9(2), pp. 323–329, 2001.
16. F. Wiedijk, *The Mathematical Vernacular,* unpublished note, available at `http://www.cs.ru.nl/~freek/notes/mv.pdf`.