Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 45–54

DOI: 10.15439/2015F229
ACSIS, Vol. 5

Equality in Computer Proof-Assistants

Adam Grabowski, Artur Korniłowicz
Institute of Informatics, University of Białystok,
ul. Ciołkowskiego 1 M, 15-245 Białystok, Poland
Email: {adam, arturk}@math.uwb.edu.pl

Christoph Schwarzweller
Department of Computer Science, University of Gdańsk,
Wita Stwosza 57, 80-952 Gdańsk, Poland
Email: schwarzw@inf.ug.edu.pl

***Abstract*—Equality is fundamental notion of logic and mathematics as a whole. If computer-supported formalization of knowledge is taken into account, sooner or later one should precisely declare the intended meaning/interpretation of the primitive predicate symbol of equality. In the paper we draw some issues how computerized proof-assistants can deal with this notion, and at the same time, we propose solutions, which are not contradictory with mathematical tradition and readability of source code. Our discussion is illustrated with examples taken from the implementation of the MIZAR system.**

I. INTRODUCTION

THE ROLE of equality in mathematics is indispensable. Linear equations represented and accompanied by graphs seem to be one of the primary mathematical exercises for children, where also recognizing which objects are identical is important. Finding solutions for systems of equations [2], formulas for calculating integrals, values of various mathematical functions, etc., is the basic mathematical activity all engineers are familiar with. Hence it is not surprising that equational provers were one of the primary computerized tools developed for use by mathematicians, after offering simple numerical tools and methods – in fact also based on equality since they essentially handle sequences of equalities.

As a mathematical proof, a proof in a computerized system is just a sequence of proof steps, we could expect that it can be discovered automatically within a reasonable universe of discourse. Typically, a proof search explodes exponentially if the universe and/or the method is not properly chosen. Essentially the process of finding an equational proof is fairly simple. Given sets of equalities can be merged to define a substitution operation by iteration. E.g., in cases when the equalities are between terms and variables, or imply such equalities that can be derived, the substitution operation is the result of (1) (simultaneous) substitution of terms for corresponding variables, according to the given equalities, and (2) iterating this process on results. The iteration step (2) can lead to the target or fail if sooner or later the equalities became too complex to handle. The idea is to have a handy set of underlying techniques for machine learning and to restrict the field of expansions.

The results are always tempting, as after identifying two objects as equal – no matter how different they seem to have been – all knowledge about one of them applies to the other also. Moreover, from the moment of identifying objects, it does not matter with which of them we are dealing, properties discovered for one of them automatically carry over to the other one. Thus, identifying objects as equal can make certain work (e.g., in some non-procedural or extensional work) just easier. Depending on the mathematical context or computational environment, the use of the quality predicate may not be so simple as they look, and saying that two objects are equal we often mean certain level of *abstraction* which is the core of mathematics.

Informally the notion of equality is clear, but following [19] we cannot be sure about that:

> *... when it comes to a crisis of rigorous argument, the open secret is that, for the most part, mathematicians who are not focussed on the architecture of formal systems per se, mathematicians who are consumers rather than providers, somehow achieve a sense of utterly firm conviction in their mathematical doings, without actually going through the exercise of translating their particular argumentation into a brand-name formal system.*

Developers of computerized proof-assistants must be prepared for such exercises, furthermore – they have to provide tools for solving them. Our paper is a result of some thoughts presented in [7] and [3]. However, we are focusing not on computer algebra systems as in [7], and also not on the role of equality in mathematical education (although mathematical proof-assistants are definitely useful in this area). We started from the place where [7] posed some important questions: the area where automated reasoning and computer algebra can interact, and we went ahead of discussions, focusing on real-life implementations of the theoretical ideas: how the automatic proof-checker can cope with the predicate of mathematical equality and corresponding predicate symbol of equality. For our work in this paper, we have chosen MIZAR proof-checker described in [1]. The MIZAR system is much closer to automated theorem prover, although some basic elements of computer algebra are also implemented. In this sense this discussion is a kind of a counterpart of Davenport's research from the viewpoint of MIZAR proof-checker.

Essentially, all with the exception of the very basic MIZAR examples are of our authorship: the first author is mainly responsible for the formalization of lattice theory, rough and fuzzy sets [26], [37], the second author's work is on hard-coding of new MIZAR constructions (e.g. reductions), while the third author delivered examples from abstract algebra. We

978-83-60810-66-8/$25.00

hope then that our paper is much more than a theoretic discussion on selected issues on the usage of equality predicates.

The structure of the paper is as follows. At the beginning we draw some initial remarks on the properties of the equality and show how one can solve equational problems with the help of a computer, not necessarily in a fully readable way. Then we focus on the MIZAR system, explaining the implementation of the equality both from purely logical point of view, and extensionally, in set theory (including two extensions: rough and fuzzy sets). Section VI starts the discussion on specific implementation issues in MIZAR: analysis of terms, properties, and built-in computations. In Section XII, based on the concrete example, we show how the discussed techniques work to find the compromise between readability and writability, and then we present the statistics of the use of described constructions in MIZAR Mathematical Library (MML). The final part is devoted to the discussion of the structural understanding of the equality, where injections and isomorphisms are typically used by mathematicians.

II. EQUATIONAL CHARACTERIZATION

The importance of equational characterization is obvious – varieties are defined in this way. Among equationally defined classes of algebras, we can find all well-known problems solved with the help of powerful provers, with the Robbins problem [21] (the alternative set of axioms for Boolean algebras) and its solution by EQP/OTTER as the most prominent example. And even if Robbins problem's importance for Boolean algebras is not crucial, this specific set of axioms is quite interesting. E.g., we can point out many problems in lattice theory (presented as short equational bases) solved as a by-product of this computerized system achievement [9].

Absolute equality has the following properties defining it as equivalence relation between the objects in the considered universe of discourse (i.e., domain of objects):

- reflexivity,
- symmetry, and
- transitivity.

Furthermore, the absolute equality should satisfy the so-called Leibniz's Law (or identity of indiscernibles), which is a kind of closure with respect to properties or substitution property.

> *Two objects x and y are equal if for every predicate P we have $P(x)$ if and only if $P(y)$, that is, x and y are equal if they cannot be distinguished using predicates.*

This is an intuitively clear definition, however hard to check, in particular in proof assistants – one can ask *which* universe of discourse should be taken. On the other hand, in some sense, the equality can be treated just like ordinary predicate.

III. AUTOMATION – THE INITIAL APPROACH

If we are not aware of all fears of the precise meaning and all shades of mathematical equality, we can see its use with computer knowledge systems very flawlessly. We can use equational provers e.g. in the area of lattice theory (as EQP/Prover9 was very successful in this domain), which gives representation of various mathematical areas – topology, algebra, logic, geometry, etc.

Remembering that lattices are structures

$$\langle L, \sqcup, \sqcap \rangle$$

where both binary operations $\sqcup$ and $\sqcap$ are commutative, associative, and satisfy the absorption laws, given as axioms, we can obtain

$$a \sqcup a = a$$

as a result of the six axioms. Essentially, the easy proof (sometimes credited to Dedekind) doesn't need them all, although caused some confusion (e.g., early versions of lattice axiomatics included both idempotences as additional axioms). We can easily obtain a proof using any equational prover. Let us stick to our favourite Prover9 – the direct successor of OTTER:

```
formulas(assumptions).
x ^ y = y ^ x.
x ^ (y ^ z) = (x ^ y) ^ z.
x ^ (x v y) = x.
x v y = y v x.
x v (y v z) = (x v y) v z.
x v (x ^ y) = x.
end_of_list.
```

pushing all the axioms into the assumptions and the desired equality into the goals:

```
formulas(goals).
x v x = x.
end_of_list.
```

and after a while one can get the answer, which is maybe not very readable for a human, but definitely assures us that the proven formula is true.

```
1   x v x = x.          [goal].
5   x ^ (x v y) = x.    [assumption].
9   x v (x ^ y) = x.    [assumption].
10  c1 v c1 != c1.      [deny(1)].
22  x v x = x.          [para(5(a,1),9(a,1,2))].
23  $F.                 [resolve(22,a,10,a)].
```

Even for the reader unfamiliar with Prover9 syntax [20] it is clear that only two axioms are really needed. Bigger proofs are not that readable and additional transformations are useful to show the essence of the proof. In the column on the right hand side in square braces one can note the so-called *tactics* used in the proof search – among assumptions and goals, `deny` denotes the denial of the goal (Prover9 uses proofs by contradiction, then paramodulation and resolution is done). Essentially, the more readable counterpart of the proof given by Prover9 is the following *proof object* which contains concrete proof steps.

```
(5 (input) (= (meet v0 (v v0 v1)) v0) NIL)
(9 (input) (= (v v0 (meet v0 v1)) v0) NIL)
(10 (input) (not (= (v (c1) (c1)) (c1))) NIL)
(24 (instantiate 5 ((v0 . v100)))
```

```
     (= (meet v100 (v v100 v1)) v100) NIL)
(25 (instantiate 9 ((v0 . v100)
     (v1 . (v v100 v1))))
      (= (v v100 (meet v100 (v v100 v1))) v100)
       NIL)
(26 (paramod 24 (1) 25 (1 2))
     (= (v v100 v100) v100) NIL)
(22 (instantiate 26 ((v100 . v0)))
     (= (v v0 v0) v0) NIL)
(27 (instantiate 22 ((v0 . (c1))))
     (= (v (c1) (c1)) (c1)) NIL)
(23 (resolve 27 () 10 ()) false NIL)
```

IV. The Mizar System

Formalization of mathematics is a practise of mathematics, or specific mathematical activity, by using a formal language suitable for computerized systems [36]. Here, we mean that a practise of mathematics means acts of proving theorems and correctness of definitions according to classical logic and Zermelo-Fraenkel set theory. Such activity, obviously without the use of computers, is dated back to Peano and Bourbaki, and typically, all areas of mathematics use specific, more-or-less formal languages. Computer certification of mathematics can be useful for many reasons – computers open new possibilities of information analysis and exchange, they can help to discover new proofs or to shed some light on approaches from various perspectives. With the help of such automated proof assistants one can observe deeper connections between various areas of mathematics.

Even if formalization of mathematics (even outside any computerized assistants) could potentially depend only on specific logical layer, set-theoretical counterpart is often, or typically, indispensable in practical applications (of course, one can imagine expressing e.g. Zermelo theorem in terms of pure predicates as Rasiowa and Sikorski mentioned in their *The Mathematics of Metamathematics* [30], but we aim at practically useful programs). Essentially then, alongside with logical connectives, e.g. equivalence, a mathematical system can include equality relation or predicate. Its characteristic properties are those commonly accepted in equivalential logics – reflexivity, symmetry, and transitivity, as we mentioned before.

As the testbed of our considerations we have chosen the Mizar system, developed by the team we are members of. It was created in the early 1970s in order to assist mathematicians in their work. Now the system consist of three main parts: (1) a special formal language, in which mathematics can be expressed. This language is close to the vernacular used by human mathematician [14]. When used, the language expressions can be automatically checked for grammatical correctness; (2) the software, which verifies the correctness of formalized knowledge, in the classical logical framework; and last, but not least, (3) a huge collection of certified mathematical knowledge – the MML.

Here, we can quote the source written in Mizar file `HIDDEN` containing basic built-in properties of primitives – one can see reflexivity and symmetry for the equality. Unfortunately, there is no *transitivity* property implemented for predicates in Mizar, hence it was hard-coded by the developers of the system.

```
definition let x,y be object;
  pred x = y;
  reflexivity;
  symmetry;
end;
```

In fact, as `HIDDEN` is one of the two axiomatic files in the MML (the second one is `TARSKI`, which will be mentioned in the next section), there are no proofs for these properties. The formal language in our examples with Mizar is pretty close to the every-day language in ordinary mathematics. For precise syntax details, we refer to [1].

V. Set-theoretic Equality and Its Extensions

If we have primitives for chosen set theory, namely the notion of a set and a primitive set-theoretic membership predicate $\in$, we can express the equality of sets in terms of $\in$ and logical connectives (including quantifiers). Clasically, it is done via extensionality.

$$\forall_X \forall_Y (X = Y \Leftrightarrow (\forall_x (x \in X \Leftrightarrow x \in Y)))$$

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

The implication in the opposite direction is provided by the implementation of the system. But obviously, this equality predicate is by no means *new* one. Of course, the above theorem can be read as the form of Leibniz's Law.

Although the notion of the equality of objects is quite concrete, its specific realizations are strongly dependent on how the object is mathematically defined. An illustrative example could be the notion of a rough set [26] treated formally in [8]. On the one hand, rough sets can be defined as the classes of equivalence with respect to a certain relation (it's really meaningless here that numerous generalizations are described in the literature, and partitions are hardly used nowadays in real-life applications of rough set theory). In such approach, two rough sets are equal if the families of subsets are equal (taken componentwise). Thus, we can define the alternative rough equality as below. It is coherent with the equality of ordered pairs, but not necessarily with families of subsets.

```
definition
  let A be Approximation_Space,
      X, Y be Subset of A;
  pred X _=^ Y means
:: ROUGHS_1:def 16
  LAp X = LAp Y & UAp X = UAp Y;
  reflexivity;
  symmetry;
end;
```

Note that the symbol of the rough equality predicate is not the ordinary =, but _=^ denoting the equality of lower and upper approximations. Earlier, we introduced similar predicates in cases of single approximations only [13], [10].

One can consider another approach, credited to Iwiński [16] – rough sets can be viewed as a pair of approximation operators – the lower and the upper one. Within a fixed approximation space, two sets are equal if both approximations are equal (from the very foundational part we can quote Kuratowski's definition of an ordered pair as

$$(a, b) = \{\{a, b\}, \{a\}\}),$$

but majority of mathematicians explore an usual equality of underlying elements.

Here we only mention that even if rough and fuzzy sets have much in common, the equality of fuzzy sets in MIZAR is not that harmless. Because in MML fuzzy sets are corresponding membership functions, their equality is just set-theoretic equality. For formal approach to fuzzy sets, see [11].

VI. Equational Calculus in Mizar

Equational calculus is mainly performed by a dedicated module EQUALIZER which computes the congruence closure over the collection of equalities accessible at a given inference, where the congruence closure of a relation R defined on a set A is the minimal *congruence relation* (a relation that is simultaneously reflexive, symmetric, transitive, and compatible) containing the original relation R. In other words, R is congruence iff it is an equivalence relation satisfying

$$\forall_{x_1,x_2,y_1,y_2 \in A}(x_1, x_2) \in R \Leftrightarrow (y_1, y_2) \in R.$$

The computed congruence closure is used by the MIZAR CHECKER to detect a possible contradiction and to refute the inference (MIZAR is a disprover; by using a technique similar to adding the negation of the goal to the list of available premises can be observed in line 10 of the Prover9 proof in Section III), which can happen if one of the following cases holds

- there are two premises of the form $P[x]$ and $\neg P[y]$ and x, y are congruent, or
- there is a premise of the form $x \neq y$ when x, y are congruent;

where two elements x and y are congruent, if the pair (x, y) belongs to the congruence closure.

There are many possible sources of equalities, which can be taken into account during the analysis of a given inference. They can be grouped into the following categories:

- occurring explicitly in a given inference,
- term expansions (`equals`),
- properties,
- term reductions,
- term identifications,
- arithmetic,
- type changing (`reconsider`),
- others, e.g. processing structures.

Some of them are of very basic character (e.g. arithmetic), two last ones are extremely dependent on the MIZAR language specification of objects (in this case, structures and type changing statements), but the remaining five are of very general character, and we can easily remap them with the ordinary human reasoning.

The following is an example of a very basic case of equalities, which are stated in the statement to be proved, e.g.:

```
theorem
  for a,b being Element of INT.Ring
    for c,d being Integer st a = c & b = d
      holds a + b = c + d;
```

In the following section(s), we give examples of equalities for the listed categories, at least for those that are not-so-intuitively clear.

VII. Term Expansions and Term Reductions

There are two MIZAR strategies based on substitutions – they act dually in a sense that one of them increases the length of the formula (measured by the number of characters), the other goes in the opposite direction preventing a little bit from uncontrolled growth of terms.

A. Expansions

One of the methods for defining new functors can use the following syntax:

```
definition
  let x_1 be θ_1, x_2 be θ_2, ..., x_n be θ_n;
  func ⊗(x_1,x_2,...,x_n) -> θ equals :ident:
    τ(x_1,x_2,...,x_n);
  coherence
  proof
    thus τ(x_1,x_2,...,x_n) is θ;
  end;
end;
```

which introduces a new functor $\otimes(x_1, x_2, \ldots, x_n)$ which is equal to $\tau\,(x_1, x_2, \ldots, x_n)$. Such definitions, whenever terms $\otimes(x_1, x_2, \ldots, x_n)$ occur in an inference, allow the VERIFIER to generate equalities

$$\otimes(x_1, x_2, \ldots, x_n) \;=\; \tau\,(x_1, x_2, \ldots, x_n)\,.$$

For example,

```
definition
  let x,y be Complex;
  func x - y -> Complex equals
    x + (-y);
  coherence;
end;
```

causes that all instantiations of terms `x-y` are expanded to `x+(-y)`. As a gain of such expansions, for example, the equality `a-b-c = a+(-b-c)` is a direct consequence of associativity of addition. It holds because the term `a-b` is expanded to the term `a+(-b)`, and the term `a-b-c` is expanded to the term `a-b+(-c)`, and both give `a+(-b)+(-c)`. On the other hand, the term `-b-c` is expanded to the term `-b+(-c)`, which creates the term `a+(-b+(-c))`, that is, the associative form of `a+(-b)+(-c)`. An important feature of this kind of term expansions is that it is "a one-way" expansion, in the sense, that terms $\otimes(x_1, x_2, \ldots, x_n)$ are

expanded to $\tau\,(x_1, x_2, \ldots, x_n)$, but not vice-versa. The reason of such treatment is to avoid ambiguity of expansions and over-expanding terms.

B. Term Reductions

Another method of imposing the EQUALIZER to generate extra equalities based on terms occurring in processed inferences are *term reductions*, presented in [17], with the following syntax:

```
registration
  let x_1 be θ_1, x_2 be θ_2, ..., x_n be θ_n;
  reduce τ_1(x_1, x_2, ..., x_n) to τ_2(x_1, x_2, ..., x_n);
  reducibility
  proof
    thus τ_1(x_1, x_2, ..., x_n) = τ_2(x_1, x_2, ..., x_n);
  end;
end;
```

Term reductions can be used to simplify terms to their proper parts of terms (sub-terms). This simplification relies on matching terms existing in the processed inference with left-side terms of all accessible reductions, and whenever the EQUALIZER meets an instantiation σ of the term $\tau_1(x_1, x_2, \ldots, x_n)$, it makes σ equal to its sub-term equivalent to $\tau_2(x_1, x_2, \ldots, x_n)$.

The restriction about simplifying terms to their proper subterms, not to any arbitrarily chosen terms, is to fulfill the general rule established for the system, that the EQUALIZER does not generate extra terms and does not expand the universe of discourse.

An example of a possible reduction could be reducing the first power of a complex number to the number (`c` is a subterm of `c|^1`).

example

```
registration
  let c be Complex;
  reduce c|^1 to c;
  reducibility;
end;
```

Reducing the zero power of a number to one is not allowed (`1` is not a sub-term of `c|^0`) – as in the source code below – as a result the following error will be output in a comment ("`::`" starts a comment).

prohibited use of a reduction

```
registration
  let c be Complex;
  reduce c|^0 to 1;
::>          *257
  reducibility;
end;

::> 257: Right term must be
::>      a sub-term of the left term
```

Reductions are recently implemented and their impact on the library is not very big yet. First of all, we can imagine even very complicated reductions, which can be useful in very exceptional cases. The main aim is to reflect human reasoning rather than to force unusual calculations, even if they are straightforward for the machine.

VIII. PROPERTIES

Properties in MIZAR are special formulas, which can be registered while defining functors, see [1], [23]. MIZAR supports `involutiveness` and `projectivity`, for unary operations, and `commutativity` and `idempotence`, for binary operations. If a property is registered for some functor, the EQUALIZER processes appropriate equalities adequate to the property, where for `involutiveness` the equality is

$$f(f(x)) = x,$$

for `projectivity` it is

$$f(f(x)) = f(x),$$

for `commutativity` it is

$$f(x, y) = f(y, x),$$

and for `idempotence`

$$f(x, x) = x.$$

IX. TERM IDENTIFICATIONS

In mathematics, there are different theories, which at some of their parts are about the same objects. For example, when one considers complex numbers and extended real numbers (reals augmented by $+\infty$ and $-\infty$) and discusses basic operations on such numbers (like addition, subtraction, etc.), it can be quickly recognized that if the numbers are reals, the results of these operations are equal to each other. That is, there is no difference, if one adds reals in the sense of complex numbers or in the sense of extended real numbers. Therefore, pairs of such operations could be identified on appropriate sets of arguments.

MIZAR provides a special construction for such identifications [4] with syntax:

```
registration
  let x_1 be θ_1, x_2 be θ_2, ..., x_n be θ_n;
  let y_1 be Ξ_1, y_2 be Ξ_2, ..., y_n be Ξ_n;
  identify τ_1(x_1, x_2, ..., x_n) with τ_2(y_1, y_2, ..., y_n)
   when x_1 = y_1, x_2 = y_2, ..., x_n = y_n;
  compatibility
  proof
   thus x_1 = y_1 & x_2 = y_2 & ... & x_n = y_n
    implies τ_1(x_1, x_2, ..., x_n) = τ_2(y_1, y_2, ..., y_n);
  end;
end;
```

and, whenever the EQUALIZER meets an instantiation σ of the term $\tau_1(x_1, x_2, \ldots, x_n)$, it makes σ equal to the appropriate

instantiation of $\tau_2(y_1, y_2, \ldots, y_n)$. A gain of using such identifications is that all facts proven about $\tau_2(y_1, y_2, \ldots, y_n)$ are applicable for $\tau_1(x_1, x_2, \ldots, x_n)$, as well.

An example of identification taken from the MIZAR MATHEMATICAL LIBRARY could be the lattice of real numbers with operations `min`, `max` as the infimum and supremum, respectively, of two elements of the lattice, see [5].

```
registration
  let a,b be Element of Real_Lattice;
  identify a "\/" b with max(a,b);
  compatibility;
  identify a "/\" b with min(a,b);
  compatibility;
end;
```

By having such identifications declared, i.e., registered, as `registration`, reasonings about the lattice operations can automatically use facts about real numbers. For example, the associativity of the supremum is a direct consequence of the associativity of the maximum: `max(max(a,b),c) = max(a,max(b,c))`

A less obvious example of such term identifications is connection of lower and upper approximations of rough sets with the topological interior and topological closure, respectively, see [12]. The problem is that the topological closure coincides with the notion of the upper approximation (both possess Kuratowski closure's properties), but topological spaces and approximation spaces are formally distinct structures. We can lift both into some other common one.

```
registration
  let T be with_equivalence
    naturally_generated non empty TopRelStr;
  let A be Subset of T;
  identify LAp A with Int A;
  identify UAp A with Cl A;
end;
```

The latter registration would allow for mixed use of the lower approximation instead of interior operator and vice versa. Here the mathematician's understanding of the identification is as isomorphism (or an analogon) instead of equality.

X. BUILT-IN COMPUTATIONS

Another source of equalities processed by the EQUALIZER are special built-in procedures for processing selected objects. Generating equalities by these routines is controlled by the environment directive **requirements**, see [23]. In our interest are two procedures dealing with boolean operations on sets (`BOOLE`) and basic arithmetic operations on complex numbers (`ARITHM`).

A. Requirements *BOOLE*

```
X \/ {} = X;    X /\ {} = {};    X \ {} = X;
{} \ X = {};    X \+\ {} = X;
```

B. Requirements *ARITHM*

```
x + 0 = x;    x * 0 = 0;    1 * x = x;
x - 0 = x;    0 / x = 0;    x / 1 = x;
```

Moreover, requirements `ARITHM` provides procedures for solving systems of linear equations over complex numbers.

XI. TYPE CHANGING STATEMENTS

It is quite common situation, when one object can be treated as an element of different theories or different structures. For example, the empty set is the empty set in set theories, but it is also the zero number is some arithmetics. In computerized mathematics, to allow systems to distinguish and understand symbols clearly and to avoid ambiguity, it is often required to express types of objects explicitly.

MIZAR provides a special rule (**reconsider**) for forcing the system to treat a given term as if its type was the one stated.

For example, to consider the number 0 as an element of the field of real numbers (for example, to prove that it is the neutral element of its additive group), one can state

```
reconsider z = 0 as Element of F_Real;
```

The equality `z = 0` is obviously processed by the EQUALIZER.

XII. EXAMPLE WITH AUTOMATIZATION

In this section we present an example how all described above techniques can automatize reasoning and make proofs shorter or even make theorems obvious. The working example (about elements of the additive group of real numbers `G_Real`) with all automatizations switched-off and all basic proof steps written within the proof is as follows:

```
theorem
  for a being Element of G_Real holds
    a + 0.G_Real = a
  proof
    let a be Element of G_Real;
    reconsider x = a as Real;
B:  0 in REAL by XREAL_0:def 1;
A:  0.G_Real = the ZeroF of G_Real
                 by STRUCT_0:def 6
      .= In(0,REAL) by VECTSP_1:def 1
      .= 0 by B,SUBSET_1:def 8;
    thus a + 0.G_Real
      = (the addF of G_Real).(a,0.G_Real)
                 by ALGSTR_0:def 1
     .= addreal.(a,0.G_Real) by VECTSP_1:def 1
     .= x + 0 by A,BINOP_2:def 9
     .= x by ARITHM:1
     .= a;
  end;
```

while the theorem is obvious when all provided mechanism are utilized.

The equality

```
  a + 0.G_Real =
    (the addF of G_Real).(a,0.G_Real);
```

is a consequence of the "equals" expansion of the definition:

```
definition
  let M be addMagma;
  let a,b be Element of M;
  func a+b -> Element of M equals
:: ALGSTR_0:def 1
  (the addF of M).(a,b);
end;
```

The equality `a + 0.G_Real = x + 0` is a consequence of the equality `x = a` (reconsider), the equality `0.G_Real = 0` and the term identification:

```
registration
  let a,b be Element of G_Real, x,y be Real;
  identify a+b with x+y
  when a = x, b = y;
  compatibility by BINOP_2:def 9;
end;
```

The equality `0.G_Real = 0` is a consequence of the "equals" expansion of the definition:

```
definition
  let S be ZeroStr;
  func 0.S -> Element of S equals
:: STRUCT_0:def 6
  the ZeroF of S;
end;
```

and the "equals" expansion of the definition:

```
definition
  func G_Real -> strict addLoopStr equals
:: VECTSP_1:def 1
  addLoopStr (# REAL,addreal,In(0,REAL) #);
end;
```

and the term reduction:

```
registration
  let r be Real;
  reduce In(r,REAL) to r;
  reducibility;
end;
```

The equality `x + 0 = x` is a consequence of built-in calculations over complex numbers. Finally, the equality `x = a` is a trivial consequence of the "reconsider".

What is especially useful, the distribution of MIZAR contains programs which detect if the reference (or the proof step) is really necessary for the checker. Alternatively, we can test by brute force if the construction is useful in a specific case: adding an environment directive to the preamble of the article [25] and call the cleaning utilities; if no changes will be done, the directive is deleted. Of course, there is no need to add all directives to all articles (some of them can be just from another area of mathematics), note also that even useful constructions, say expansions, can significantly slown down the proof checking by expanding the universe of discourse.

An illustrating example was the splitting of the equality of sets into two inclusions: it is useful, but even if, according to the von Neumann construction, all natural numbers are sets, the expansion of the equality of numbers into two inclusions isn't especially feasible (even if mathematically reasonable, but who would prove inclusions like $2 \subseteq 3$, perhaps besides the person interested in the arithmetic of ordinals?).

Every MIZAR article contains a preamble with the list of files from MML which will be used. There are 10 keywords for that (and respectively, 10 environment directives). We focus only on those tightly connected with our paper, mainly of three items:

- registrations, responsible for various registrations of clusters, including reductions and identifications;
- equalities, which allow to expand definitions given by `equals`;
- requirements, which turn on (mainly arithmetical or set-theoretic) calculations.

With all the automatics switched on, one can obtain the formula from the third section

```
a "\/" a = a;
```

as a result of declared reduction (or alternatively, the `idempotence` property). This however cannot be enough for a human user who wants to see what is really going on here, i.e.:

```
a "\/" a = (a "/\" (a "\/" a)) "\/" a
                              by LATTICES:def 9
        .= a by LATTICES:def 8;
```

where both definitions are actually axioms – the absorption laws defined in the form of attributes.

Of course, the equality can be introduced in a non-explicit way as in the examples of problems of purely equational character. Remember that one can, apart from the above considerations, register the following cluster:

```
registration
  let X be set;
  cluster X \ X -> empty;
  coherence;
end;
```

which is effectively equivalent to

```
  X \ X = {};
```

adding it automatically as one of the premises to every inference where applicable. Remember that in order to assure the needed identification, one should not use the reduction ($\emptyset$ is not a sub-term of `X \ X`).

```
registration
  let T be TopSpace,
      A be 1st_class Subset of T;
  cluster Border A -> empty;
end;
```

To quote less straightforward example, in the area of rough sets (expressed in Isomichi style) it can be the identification of first class subsets with crisp sets, consequently their borders are the empty sets as the set-theoretic difference $A \setminus A$. The explanation and the details of the corresponding implementation of rough sets can be found in [13].

XIII. SOME STATISTICS

We illustrate our discussion by the number of concrete constructions used within the MML by Table I.

Table II shows how often selected properties are declared (and proved) in MML. The numbers however can be more or less accidental: authors can just omit a property when developing some theories, and can state it in a form of explicit formula, or just the proof of it can be too complex. In Table

Table I
OCCURRENCES OF BASIC CONSTRUCTIONS IN MML

Construction	keyword	occurrences
type changing statements	`reconsider`	66115
equalities	`equals`	3678
identifications	`identify`	172
reductions	`reduce`	139

Table II
OCCURRENCES OF PROPERTIES IN MML

Property	occurrences
`commutativity`	150
`idempotence`	18
`involutiveness`	36
`projectivity`	20

III, we listed how many times in MIZAR articles (among 1240 in total in the whole MML) selected properties were used. We have chosen among three areas: basic set theory, arithmetic of complex numbers, and some properties of pairs. Even if MML is based on ZFC set theory, the calculations are used very extensively.

Here we can point out the scale we refer to. There are about 11 thousand definitions and 55 thousand theorems proven in MML. This is contained among 2 million lines of code (about 90 MB of mathematical texts). One can ask a question why the commutativity of the set-theoretic union is used less often than that of set-theoretic intersection. Similar (although dual) issue with arithmetic is clear: the complex addition is just used more frequently. Symmetric difference $X \dot{-} Y$ is just of marginal interest and, which can be also a kind of explanation after deeper research, it is defined as equal to the combination of the other operations, namely $X \dot{-} Y = (X \setminus Y) \cup (Y \setminus X)$, where the commutativity of $\cup$ (understood automatically) can do some useful work.

XIV. VARIOUS MATHEMATICAL EQUALITIES

Equality in mathematics has many facings; of course there are the usual identities $x = y$, such as for example $2+1 = 3$ or $(x+1)^2 = x^2+2x+1$. Note however, that even in these cases equality depends on the domain we are working in: $2+1 = 3$ is true in $\mathbb{Z}$ but not in $\mathbb{Z}_2$. The second equation can be seen as an equation between (real) numbers or between polynomials (not necessarily) over the real numbers.

Table III
THE NUMBER OF CONCRETE USES OF PROPERTIES

Property	No. of Mizar articles	No. of implicit uses
$X \cup Y = Y \cup X$	483	4721
$X \cap Y = Y \cap X$	547	6232
$X \dot{-} Y = Y \dot{-} X$	7	57
$a + b = b + a$	670	9548
$ab = ba$	429	5734
$\{x, y\} = \{y, x\}$	118	857

There are other kinds of equality. Among the most important ones is identifying objects of a given structure with respect to a certain property. Formally, this is described by an equivalence relation r, for example $r_p(x, y)$, if x and y are integers such that x mod p = y mod p. This in particular is applied when, for example, constructing $\mathbb{Z}_p$, the integers modulo p: All integers leaving the same remainder when dividing them by p are identified using the equivalence relation r_p. Formally, equivalence classes with respect to r_p are built, and these are the elements of $\mathbb{Z}_p$ – on which the usual operations are defined. Using this construction $\mathbb{Z}_p$ does not contain any numbers, this is later achieved by "identifying an equivalence class with a number", so that $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ is used. Note, however, that this formally also means "changing the operations of $\mathbb{Z}_p$ appropriately" to match with elements of $\{0, 1, \ldots, p-1\}$.

In some sense this last step – going from equivalence classes to non-negative integers smaller than p – can be seen as changing the representation of the integers modulo p. In fact it does not matter whether we consider $\mathbb{Z}_p$ as a set of equivalence classes or as $\{0, 1, \ldots, p-1\}$. Aside from the names of the objects, the two structures are the same: The objects behave the same way, that is, the effect of the operations is identical.

This in fact is the most important kind of mathematical equality: The two structures are isomorphic, that is there exists a (bijective) function i from $\mathbb{Z}_p$ to $\{0, 1, \ldots, p-1\}$ respecting the domain's operations, i.e. $i(x + y) = i(x) + i(y)$, and similarly for all other operations of $\mathbb{Z}_p$. From a mathematical point of view, this completely defines an equality in the sense of the first section: Properties discovered for one of the structures automatically carry over to the other one. In a proof-assistant, this however is not that obvious. First of all, both structures must have been constructed in the prover to describe the isomorphism. This is not always the case and is often technical and tedious, or impossible. Moreover, by having a property/theorem for one of the isomorphic structures, usually some work remains to be done in order to have the same property/theorem for the other one. For example $\mathbb{Z}_p$ is a field, if p is prime. Proved in one of the isomorphic structures, carrying it over to the other one requires to translate the property with the help of the isomorphism i. Here of course the theorem that $i(x*y) = i(x)*i(y)$, where y is the inverse of x does the trick. The interesting discussion from the category theorist's point of view is given in [19].

Observe that our discussion here is not in contradiction with the Leibniz's Law mentioned at the beginning of our paper. Such an identity of indiscernibles does not coincide with equality via isomorphisms. For example, a field is prime, if it does not have any proper subfield. One could now expect that two isomorphic fields are either both prime or not. This, however, is not the case as the isomorphic fields $K(X)$ and $K(X^2)$ show. We will therefore mostly deal with equality via isomorphisms.

XV. Data Structure Equality

From a computer scientist's view, the equality is not that easy as Leibniz's Law could suggest. Almost immediately one observes that, e.g. in a programming language, it is quite a difference whether the objects to be equal are just plain numbers or more structured objects such as arrays or trees. For short, equality, and consequently its handling, heavily depends on the object's types. In programming languages, of course, there exist a number of techniques allowing users to exactly define equality of objects of a given type, e.g. via type classes in Haskell [32]. In general, intelligent systems should be able to handle equalities of different kinds, even delivered by external tools [24].

Proof assistants dealing with the notion of a mathematical proof, therefore, should provide means for equalities occurring in mathematics. In mathematics, however, the notion of equality is much more elaborated. To give an easy example, two functions f and g are equal if their domain D and codomain C are equal, and if $f(x) = g(x)$, for all $x \in D$. Of course, this apparent incoherence can be easily explained considering natural hierarchy of notions many mathematicians don't use in their proofs. For example, in typical set-theoretic approaches, functions are just binary relations of a special type, and relations are subsets of the Cartesian product, which is a set of corresponding ordered pairs. Here equality takes into account not only two/four additional sets, but also equality of a – possibly infinite – number of objects of another type.

When it comes to structures, such as e.g. groups, fields or topologies, equality becomes even more sophisticated. This is not only due to the growing complexity of the objects; mathematicians have developed a special style of informal handling with various kinds of equality. The point is that, in proof assistants, each kind of equality between objects of a given type has to be formally defined, in order to prove that two objects of that type are equal.

XVI. Isomorphisms and Inclusions

For a given field F, one can build the ring of polynomials $F[X]$. Actually, while the construction applies to arbitrary rings, our formalization work, however, concentrates on field theory, so we restrict ourselves to fields here. Elements of $F[X]$ – polynomials over F – are basically functions from the natural numbers into F, hence obviously different from elements of F. Consequently, $F \subseteq F[X]$ cannot hold from a formal point of view. In particular one cannot prove this with the ordinary definition of $\subseteq$. Even if this would be possible, note that $F \subseteq F[X]$ here means not that F is a subset of $F[X]$, but that F is a subfield of $F[X]$, that is addition and multiplication of $F[X]$ are taken into account. Nevertheless, [35] states the following

> *We regard $F \subseteq F[X]$ by identifying the element $a \in F$ with the constant polynomial $a \in F[X]$, and observe that under this identification the units of $F[X]$ are precisely the nonzero elements of F.*

What is hidden in this statement, is the application of a mapping i sending an element $a \in F$ to the constant polynomial $p(X) = a$. The mapping i is an isomorphism between F and its i-image in $F[X]$ (which is a subfield of $F[X]$). Thus, identifying a with $a(x)$ actually means replacing the image of i with F, which then formally gives an isomorphic copy C of F under the isomorphism i, such that $C \subseteq F[X]$:

```
theorem
  for F being Field
   ex R being Ring st
     R, Polynom-Ring F are_isomorphic &
       F is a Subfield of R;
```

and

```
theorem
  for F being Field,
      a being Element of F holds
    a|F is Unit of Polynom-Ring F
      iff a is non zero;
```

This is nice and handy, because it allows to work with the easier objects of F when appropriate. In a proof assistant, however, it gives rise to quite a number of additional work and theorems: one has to define i.

Working with polynomials, however, is not the same in isomorphic polynomial rings, even if we restrict ourselves to polynomial rings over isomorphic fields. One has to apply the isomorphism i to translate from one into the other. Note, that formally $i \circ p$ is an extension of i transforming $p \in F[X]$ into a polynomial over E, that is i is applied to all of p's coefficients:

```
theorem
  for F, E being Field,
      p being Polynomial of F,
      x being Element of F,
      i being Isomorphism of F,E holds
    i(p(x)) = (i p)(i x);

theorem
  for F, E being Field,
      p being Polynomial of F,
      x being Element of F,
      i being Isomorphism of F,E holds
    p(x) = 0.F iff (i p)(i x) = 0.E;
```

Things can get even worse. The quotient field $F[X]/ < p(X) >$, for a irreducible polynomial p, formally consists of equivalence classes $[q(X)]_{<p(X)>}$, where $q(X) \in F[X]$ – this is the field in which $p(X)$ has a root. However, in [35], after showing that $F[X]/ < p(X) >$ is a field, the author states:

> *... to give an explicit description of the field $E = F[X]/<p(X)>$. Let $p(X) \in F[X]$ be an irreducible polynomial of degree n, $p(X) = a_n X^n + \cdots + a_0$. By taking representatives of equivalence classes we may regard*
>
> $$E = \{g(X) \in F[X] \mid \textit{deg } g(X) < n\}$$
>
> *as a set. Addition in E is the usual addition of polynomials, while multiplication in E ...*

Actually, this means only that E with the new defined operations is isomorphic to $F[X]/<p(X)>$. Afterwards, the following consequence is drawn in [35]:

> *If $\pi : F[X] \longrightarrow F[X]/<p(X)>$ is the canonical projection, then $\pi|_F$ is an injection, and using this we also regard $F \subseteq F[X]/<p(X)>$.*

XVII. Conclusion

Even if mathematicians have developed and used some human "mechanisms of obviousness", for hundreds of years, new technologies impose the need for new standards. Computerized systems should undoubtfully deliver the answers of some human questions and to construct models, which cannot be created by hand in a reasonable time – this is the testbed for computer algebra systems and model finders. Proof assistants, in order not to break human standards, should however support the traditional way of thinking, so there is a place for finding a reasonable balance between writability and readability of the source code, as we believe the MIZAR system can deliver. We hope that we convinced the reader that the problem of the equality treatment is harder than it looks like at the very first sight, and if automated proof-assistants are taken into account, we should take care on something more than what we called absolute equality – Davenport's data structure equality.

References

[1] G. Bancerek, C. Byliński, A. Grabowski, A. Korniłowicz, R. Matuszewski, A. Naumowicz, and K. Pąk, Mizar: state-of-the-art and beyond. In M. Kerber et al. (Eds) *Intelligent Computer Mathematics,* Lecture Notes in Artificial Intelligence, 9150, 261–279, Springer, 2015. doi:10.1007/978-3-319-20615-8_17

[2] M. Bartoszuk, Solving systems of polynomial equations: a novel end condition and root computation method. In *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, FedCSIS 2014,* M. Ganzha, L. Maciaszek, M. Paprzycki (Eds.), p. 543–552, IEEE, 2014. doi:10.15439/2014F183

[3] R. Bradford, J.H. Davenport, and C.J. Sangwin, A comparison of equality in computer algebra and correctness in mathematical pedagogy. In *Intelligent Computer Mathematics,* Lecture Notes in Computer Science, 5625, 75–89, Springer, 2009. doi:10.1007/978-3-642-02614-0_11

[4] M. Caminati, Custom automations in Mizar. *Journal of Automated Reasoning*, 50(2), 147–160, 2013. doi:10.1007/s10817-012-9266-1

[5] M. Chmur, The lattice of real numbers. The lattice of real functions. *Formalized Mathematics*, 1(**4**):681–684, 1990.

[6] Z.E. Csajbòk and T. Mihálydeák, Fuzziness in partial approximation framework. In *Proceedings of the 2013 Federated Conference on Computer Science and Information Systems, FedCSIS 2013,* M. Ganzha, L. Maciaszek, M. Paprzycki (Eds.), p. 35–41, IEEE, 2013.

[7] J.H. Davenport, Equality in computer algebra and beyond. *Journal of Symbolic Computation,* 34(4), 259–270, 2002. doi:10.1006/jsco.2002.0551

[8] A. Grabowski, Efficient rough set theory merging. *Fundamenta Informaticae,* 135(4), 371–385, 2014. doi:10.3233/FI-2014-1129

[9] A. Grabowski, Mechanizing complemented lattices within Mizar type system. *Journal of Automated Reasoning,* 55(3), 211-221, 2015, doi:10.1007/s10817-015-9333-5

[10] A. Grabowski, On the computer-assisted reasoning about rough sets. In *Monitoring, Security and Rescue Techniques in Multiagent Systems*, B. Dunin-Kęplicz, A. Jankowski et al. (Eds.), Advances in Soft Computing, 28, 215–226, Springer, 2005. doi:10.1007/3-540-32370-8_15

[11] A. Grabowski, On the computer certification of fuzzy numbers. In *Proceedings of 2013 Federated Conference on Computer Science and Information Systems, FedCSIS 2013,* M. Ganzha, L. Maciaszek, M. Paprzycki (Eds.), 51–54, IEEE, 2013.

[12] A. Grabowski, Topological interpretation of rough sets. *Formalized Mathematics*, 22(1):89–97, 2014. doi:10.2478/forma-2014-0010

[13] A. Grabowski and M. Jastrzębska, Rough set theory from a math-assistant perspective. In *Rough Sets and Intelligent Systems Paradigms,* M. Kryszkiewicz et al. (Eds.), Lecture Notes in Artificial Intelligence, 4585, 152–161, Springer, 2007. doi:10.1007/978-3-540-73451-2_17

[14] A. Grabowski and Ch. Schwarzweller, *Translating mathematical vernacular into knowledge repositories*. In: M. Kohlhase (ed.), Proceedings of the 4th International Conference on Mathematical Knowledge Management, Lecture Notes in Artificial Intelligence, 3863, 49–64, Springer Heidelberg, 2006. doi:10.1007/11618027_4

[15] G. Grätzer, *General Lattice Theory.* Birkhäuser, 1998.

[16] T.B. Iwiński, Algebraic approach to rough sets, *Bull. Polish Acad. Sci. Math.,* 35, 673–683, 1987.

[17] A. Korniłowicz, On rewriting rules in Mizar. *Journal of Automated Reasoning,* 50(2), 203–210, 2013. doi:10.1007/s10817-012-9261-6

[18] A. Korniłowicz, Definitional expansions in Mizar. *Journal of Automated Reasoning,* 55(3), 257–268, 2015. doi:10.1007/s10817-015-9331-7

[19] B. Mazur, When is one thing equal to some other thing? In *Proof and Other Dillemas: Mathematics and Philosophy,* B. Gold and R. Simons (Eds.), Spectrum, 2008.

[20] W. McCune, Prover9 and Mace4. Available at http://www.cs.unm.edu/~mccune/prover9/, 2005–2010.

[21] W. McCune, Solution of the Robbins problem. *Journal of Automated Reasoning,* 19(3), 263–276, 1997. doi:10.1023/A:1005843212881

[22] G. Moreno, J. Penabad, and C. Vázquez, Fuzzy logic rules modeling similarity-based strict equality. In *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, M. Ganzha, L. Maciaszek, M. Paprzycki (Eds.), 119–128, IEEE, 2014. doi:10.15439/2014F387

[23] A. Naumowicz, Improving Mizar texts with properties and requirements. In *Mathematical Knowledge Management 2004*, A. Asperti, G. Bancerek, A. Trybulec (Eds.), Lecture Notes in Computer Science, 3119, 290–301, Springer Heidelberg, 2004. doi:10.1007/978-3-540-27818-4_21

[24] A. Naumowicz, Automating Boolean set operations in Mizar proof checking with the aid of an external SAT solver, *Journal of Automated Reasoning,* 55(3), 285–294, 2015, doi:10.1007/s10817-015-9332-6

[25] A. Naumowicz and A. Korniłowicz, A brief overview of Mizar. In *Theorem Proving in Higher Order Logics 2009,* S. Berghofer, T. Nipkow, Ch. Urban, M. Wenzel (Eds.), Lecture Notes in Computer Science, 5674, 67–72, Springer Heidelberg, 2009. doi:10.1007/978-3-642-03359-9_5

[26] Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning about Data.* Kluwer, Dordrecht, 1991.

[27] K. Pąk, Improving legibility of formal proofs based on the close reference principle is NP-hard, *Journal of Automated Reasoning,* 55(3), 295–306, 2015. doi:10.1007/s10817-015-9337-1

[28] K. Pąk, Improving legibility of natural deduction proofs is not trivial. *Logical Methods in Computer Science,* 10(3), 1–30, 2014. doi:10.2168/LMCS-10(3:23)2014

[29] K. Pąk, Methods of lemma extraction in natural deduction proofs. *Journal of Automated Reasoning,* 50(2), 217–228, 2013. doi:10.1007/s10817-012-9267-0

[30] H. Rasiowa and R. Sikorski, *The Mathematics of Metamathematics,* Polish Scientific Publishers, Warsaw, 1963.

[31] P. Rudnicki and A. Trybulec, On equivalents of well-foundedness. *Journal of Automated Reasoning*, 23, 197–234, 1999. doi:10.1023/A:1006218513245

[32] S. Thompson, *Haskell: The Craft of Functional Programming*. 3rd ed. Addison-Wesley, 2011.

[33] A. Trybulec, A. Korniłowicz, A. Naumowicz, and K. Kuperberg, Formal mathematics for mathematicians. *Journal of Automated Reasoning,* 50(2), 119–121, 2013. doi:10.1007/s10817-012-9268-z

[34] J. Urban and G. Sutcliffe, Automated reasoning and presentation support for formalizing mathematics in Mizar. In *Intelligent Computer Mathematics*, S. Autexier et al. (Eds.), Lecture Notes in Computer Science, 6167, 132–146, Springer Heidelberg, 2010. doi:10.1007/978-3-642-14128-7_12

[35] S. Weintraub, *Galois Theory*. 2nd edition, Springer-Verlag, 2009.

[36] F. Wiedijk, Formal proof – getting started. *Notices of the AMS,* 55(11), 1408–1414, 2008.

[37] L. Zadeh, Fuzzy sets. *Information and Control,* 8(3), 338–353, 1965.

[38] S. Żukowski, Introduction to lattice theory. *Formalized Mathematics,* 1(1), 215–222, 1990.