COMPUTER RECONSTRUCTION OF THE BODY OF MATHEMATICS

Series: STUDIES IN LOGIC, GRAMMAR AND RHETORIC 18(31)

Under the Auspices of the Polish Association
for Logic and Philosophy of Science

COMPUTER RECONSTRUCTION OF THE BODY OF MATHEMATICS

Guest Editors Adam Grabowski
Adam Naumowicz

University of Białystok
Białystok 2009

Guest Editors

Adam Grabowski
Institute of Mathematics, University of Białystok
Białystok, Poland
e-mail: adam@mizar.org

Adam Naumowicz
Institute of Informatics, University of Białystok
Białystok, Poland
e-mail: adamn@mizar.org

Series: STUDIES IN LOGIC, GRAMMAR AND RHETORIC 18(31)
http://logika.uwb.edu.pl/studies/

Refereed by Andrzej Trybulec
Cover design: Krzysztof Tur

ISBN 978–83–7431–229–5
ISSN 0860–150X

WYDAWNICTWO UNIWERSYTETU W BIAŁYMSTOKU
15-097 Białystok, ul. Marii Skłodowskiej-Curie 14, tel. +48-85 745 70 59
http://wydawnictwo.uwb.edu.pl e-mail: ac-dw@uwb.edu.pl
Nakład 200 egz.

Printed in Poland by: PPHU TOTEM s.c., Inowrocław

Table of Contents

Preface

This special issue is devoted to various aspects of reconstructing the existing body of mathematics within a formal framework of a computer-based environment. Since the inception of the first systems designed for formalizing mathematics the research on automated theorem proving and proof checking has produced numerous proof-assistant tools. Their involvement in large-scale formalization projects revealed that practical formalization of whole theories, complete mathematical monographs or especially challenging proofs requires a substantial amount of dedicated work. It has become evident that for this sort of task accumulating and distributing previously formalized data is indispensable.

In this spirit the Mizar community initiated in 1989 a long-term project of building a comprehensive library of formally interrelated mathematical data, the Mizar Mathematical Library (MML) and several other proof-assistants followed. Among the collected contributions there are three articles devoted to the problems connected with maintaining such libraries, expanding them, and at the same time making them coherent, well-organized and suitable for querying. There is a presentation of the current state of the biggest digital libraries of formalized mathematics. Admittedly, it has been estimated that building a formal library to cover only undergraduate mathematics would require about 140 man-years. Therefore it has been recognized that developing a library to base further formalizations on is equally essential as improving the capabilities of proof-assistance software.

Another three articles deal with the complete process of formalizing certain proofs which starts with considering different possible approaches and finally chosing one which suits best the proof-checking system, the choice being based on known existing proofs and the contents of the library available for the proof checker. The works contain valuable observations on the underlying knowledge that is used in "normal" mathematical practice and its reconstruction in the formalization process. This reconstruction makes it possible to point out some inefficiency of the current proof-assistants as compared to the standard mathematical apparatus.

The support offered to the end-user by proof-assistants is addressed in another three articles. They describe recent enhancements in this area, ways of efficient encoding of formal data, and presenting it to the users in an attractive way following the notions used in standard mathematics. All these enhancements are highly dependent on the linguistic capabilities of a language used for formalization. A particularily valuable contribution is presented in one of the papers that proposes theoretical foundations of an alternative to the existing languages for encoding formal mathematics. The features of this proposal, which comes from within the mathematical community, can stimulate further development of existing proof tools.

Handing to the readers this special issue dedicated to the computer reconstruction of the body of mathematics, we would like to disseminate research experiences with proof-assistant systems that collect user contributions in an organized manner, attempt to build proof libraries and allow for new developments on top of previously accumulated mathematical knowledge. We believe that expanding such libraries and at the same time making them coherent and well-organized is a highly non-trivial enterprise. The collected works show that these activities are crucial for surpassing the limitations of contemporary proof-assistants.

Adam Grabowski and Adam Naumowicz

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 18 (31) 2009

A Formal Proof of Euler's Polyhedron Formula

Jesse Alama[1]

Department of Philosophy
Stanford University
alama@stanford.edu

Abstract. Euler's polyhedron formula asserts for a polyhedron p that

$$V - E + F = 2,$$

where V, E, and F are, respectively, the numbers of vertices, edges, and faces of p. This paper concerns a formal proof in the MIZAR system of Euler's polyhedron formula carried out [1] by the author. We discuss the informal proof (Poincaré's) on which the formal proof is based, the formalism in which the proof was carried out, notable features of the formalization, and related projects.

1 Euler's Polyhedron Formula

Euler first discussed his formula in a 1750 letter to Christian Goldbach:

> *Recently it occurred to me to determine the general properties of solids bounded by plane faces, because there is no doubt that general theorems should be found for them, just as for plane rectilinear figures, whose properties are: (1) that in every plane figure the number of sides is equal to the number of angles, and (2) that the sum of all the angles is equal to twice as many right angles as there are sides, less four. Whereas for plane figures only sides and angles need to be considered, for the case of solids more parts must be taken into account. [16]*

Euler does not use the term *polyhedra* but rather "solids bounded by plane faces". He goes on to enumerate some interesting propositions about polyhedra such as:

> *6. In every solid enclosed by plane faces the aggregate of the number of faces and the number of solid angles exceeds by two the number of edges, or* $F + V = E + 2$.[1]

and

[1] Euler's text has been modified to bring it into line with the notation used in this paper: he did not use the conventional English abbreviations "V", "E", and "F".

ISBN 978-83-7431-229-5 ISSN 0860–150X

> *11. The sum of all plane angles is equal to four times as many right angles as there are solid angles, less eight, that is* $4V - 8$ *right angles.*[2]

Euler expresses surprise that he has not been able to find a precedent for these relations:

> *I find it surprising that these general results in solid geometry have not been previously noted by anyone, so far as I am aware;*[3] *and furthermore, that the important ones, Theorems 6 and 11, are so difficult that I have not yet been able to prove them in a satisfactory way.*

It was not long before Euler presented his results publicly [8]. Like the letter to Goldbach, Euler's paper was programmatic: he was trying to encourage the study of three-dimensional solids as an extension of planar geometry. The "most difficult" propositions he mentioned to Goldbach were discussed in detail, though he acknowledges that his presentation does not constitute a proof. Indeed, in the preface to his paper Euler qualifies his work thus:

> *I for one have to admit that I have not yet been able to devise a strict proof of this theorem. As however the truth of it has been established in so many cases, there can be no doubt that it holds good for any solid. Thus the proposition seems to be satisfactorily demonstrated.*

Euler was not satisfied with the unfinished state of his theorem and continued working with polyhedra. Eventually he did find a satisfactory proof [7].

Perhaps because of its simplicity and elegance, many other mathematicians studied the polyhedron formula and tried to give new proofs. Cauchy, for example, connected the study of polyhedra to planar graphs: project a polyhedron onto a plane, triangulate it, and take away one triangle at a time in a way that preserves χ until only a triangle remains; we obtain the desired result $\chi = 2$ by noting that the projection with which we started "removes" a face from the polyhedron (which effectively sends one of the polyhedron's faces onto an unbounded planar region). Unlike Euler, whose conception of polyhedra was that of solid (which one can slice, as with a knife), Cauchy apparently viewed polyhedra as wireframes.

Poincaré provided a new conception of polyhedra based on incidence matrices with which he gave his own proof [22, 21] of Euler's formula.[4] Poincaré's abstract, combinatorial conception of polyhedra makes no mention of points in $\mathbf{R}^3$, nor does it come from projecting polyhedra onto a plane. Poincaré's approach even allows

[2] Euler proved that proposition 6 is equivalent to proposition 11. This is an interesting equivalence because one statement has a combinatorial flavor, while the other has an analytic flavor. Proposition 11 can be seen in the famous Gauss-Bonnet formula [27].

[3] Unknown to Euler, Descartes had actually given a proof of Proposition 11 [15]. This result of Descartes's, seems to have been missing at Euler's time; it was rediscovered in the 19th century, long after Euler's death [24].

[4] Poincaré was interested more broadly in the new subject of topology, of which he was one of the earliest explorers; his new proof of Euler's polyhedron formula was but one element in his wider topological program.

for polyhedra of arbitrary dimension; the general result[5] states that

$$\sum_{k=0}^{d-1}(-1)^k N_k = 1 + (-1)^{d+1},$$

where the integer d is the dimension of p and N_k is the number of k-polytopes of p. The classical three-dimensional version stated by Euler is obtained by setting $d := 3$. The familiar property of a polygon that the number of vertices is equal to the number of edges is obtained by putting $d := 2$. (And a 1-dimensional polyhedron is just a line segment with its two endpoints, which also falls out of the general Euler relation by putting $d := 1$.)

So far no definition of polyhedron has been given, nor have we placed any restriction on the domain of validity of Euler's relation. It is a commonplace that one has to be careful with how one defines one's terms, and the term "polyhedron" is no exception. Grünbaum writes:

> *The "Original Sin" in the theory of polyhedra goes back to Euclid, and through Kepler, Poinsot, Cauchy, and many others, in that at each stage, the writers failed to define what are the 'polyhedra'.* [13]

In addition to defining polyhedra, it is a further task to specify the domain of validity for Euler's relation to hold; it turns out that around the time of Cauchy's proof in the early 19th century, it started to become clear to mathematicians that Euler's polyhedron formula does not hold for all polyhedra. In 1811, for example, L'Hulier described "exceptions" to Euler's polyhedron formula, classifying them into three kinds. Research on polyhedra in the 19th century gradually revealed that for Euler's relation to hold one should focus on *simple connectedness*, which roughly asserts that any two vertices can be connected by a path of edges and that the faces can be continuously collapsed to a point.

(Lakatos's history [18] of Euler's polyhedron formula is an entertaining discussion of some of the historical twists and philosophical problems surrounding the result.[6])

Poincaré's definition, on which the formalization to be described is based, is probably the simplest to describe. Following Poincaré, a polyhedron is characterized by a list of *incidence matrices*, which can be understood as functions f from a cartesian product $A \times B$ of sets A and B to $\{0, 1\}$, where $f(a, b) = 1$ is understood as "a is incident with b" and $f(a, b) = 0$ is understood as "a is not incident with b". Thus to specify a polyhedron of dimension $d + 1$, one just gives d incidence matrices. Let us call such a structure an *abstract* or *combinatorial polyhedron*.

[5] Poincaré was not the first to generalize Euler's polyhedron formula to higher dimensions; that was done by L'Hullier.

[6] Indeed, a motivation for carrying out the formalization described here was to study Lakatos's philosophy of mathematics.

2 Poincaré's Proof of Euler's Polyhedron Formula

As part of his algebraic topological program, Poincaré gave a new proof of Euler's polyhedron formula. In this section we give a sketch of Poincaré's proof; for a more detailed discussion, consult Lakatos [18] (chapter 2) or Coxeter [5] (chapter 9).

First, we should say how Poincaré defines polyhedra. In his framework, a three-dimensional polyhedron is determined by five pieces of data:

- A set of vertices (the 0-polytopes),
- A set of edges (the 1-polytopes),
- A set of faces (the 2-polytopes),
- An incidence matrix that says which vertices belong to which edges, and
- An incidence matrix that says which edges belong to which faces.[7]

Conventionally there is also a 3-polytope, namely the whole polyhedron p, and we specify a new incidence matrix declaring that all faces are incident with p. Symmetrically, we conventionally define a single -1-polytope and declare that it is incident with each vertex.

More generally, a d-dimensional polyhedron is characterized by a pair $(\mathcal{F}, \mathcal{I})$ ($\mathcal{F}$ for "faces", $\mathcal{I}$ for "incidences") of finite sequences, where

- $d = \operatorname{len} \mathcal{F}$,
- $\operatorname{len} \mathcal{F} > 0$,
- $\operatorname{len} \mathcal{I} = \operatorname{len} \mathcal{F} - 1$,
- For $0 \leq n < \operatorname{len} \mathcal{F}$, we have that $\mathcal{F}_n$ is a non-empty finite set (the set of k-polytopes of p), and
- For $0 \leq n < \operatorname{len} \mathcal{I}$, we have that $\mathcal{I}_n$ is an incidence matrix for $\mathcal{F}_n$ and $\mathcal{F}_{n+1}$.

In the more general setting we again stipulate that there is one d-dimensional polytope, namely p, that is incident with all $(d-1)$-polytopes; also, we stipulate that there is one -1-dimensional polytope that is incident with all 0-polytopes.

Theorem 1. *For every simply connected polyhedron p, we have*

$$\sum_{k=0}^{d-1} N_k = 1 + (-1)^{d+1},$$

where d is the dimension of p and N_k is the number of polytopes of p of dimension k.

For a polyhedron p and an integer k, let the k-chains of p be the powerset of the set of k-polytopes of p. The k-chains of p naturally form a vector space over the two-element field F_2, where vector addition is represented by disjoint union (symmetric difference); call this space C_k. The relation between C_k and polyhedra can be seen in the fact that the dimension of C_k is precisely N_k, the number of k-polytopes of

[7] In fact, Poincaré used a single incidence matrix to represent a polyhedron. The matrix is a block matrix, two of whose blocks are just the zero matrix, expressing the fact that vertices are not (strictly speaking) incident with faces but only with edges.

p. (Reason: the singleton subsets of $\mathcal{F}_k$ are a basis for C_k.) The boundary $\partial_k c$ of a k-chain c is the $(k-1)$-chain

$$\{x \in \mathcal{F}_{k-1} \colon x \text{ is incident with an odd number of } k\text{-polytopes of } c\}.$$

In other words, a $(k-1)$-polytope x belongs to the boundary of a k-chain c iff

$$\sum_{y \in c} \mathcal{I}_{k-1}(x, y) = 1,$$

where the sum is taken modulo 2. The boundary operation ∂_k is a linear transformation from C_k to C_{k-1}. It turns out that the k-chains c whose boundary is empty (all $(k-1)$-polytopes are incident with c an even number of times) form a subspace, Z_k, of C_k. Such k-chains are called k-circuits (sometimes also called k-cycles). Another important subspace of the k-chain space C_k consists of those k-chains that are the boundary of a $(k+1)$-chain; for lack of a better name, let B_k (for "bounding") denote this subspace.

The property of *simple connectedness* is the property that $B_k = Z_k$, that the k-circuits are the bounding k-chains. The inclusion $B_k \subseteq Z_k$ says that $\partial_{k+1}\partial_k \equiv 0$. The reverse inclusion intuitively says that the only way something can be a cycle is if it "traverses" a "face". This fails in cases where, for example, a face has a hole in it (one can go around the boundary of the inner hole, but there's no face that one is traversing).

Proof of Theorem 1. If p is simply connected, then

$$Z_k = B_k,$$

so that

$$\dim Z_k = \dim B_k.$$

Since $N_k = \dim C_k$, we have by the rank+nullity theorem that

$$N_k = \dim C_k = \dim B_{k-1} + \dim Z_k = \dim B_{k-1} + \dim B_k.$$

Thus

$$\sum_{k=0}^{d-1} (-1)^k N_k = \sum_{k=0}^{d-1} (-1)^k (\dim B_{k-1} + \dim B_k) = \dim B_{-1} + (-1)^{d-1} \dim B_{d-1}.$$

The last equation follows because of the hypothesis of simple connectedness. Now $\dim B_{-1} = 1$, since B_{-1} is a two-element vector space (it contains the empty chain as well as the singleton chain containing the unique -1-polytope). And $\dim B_{d-1} = 1$ for the same reason: it contains the empty chain as well as the "full" chain containing all the $(d-1)$-polytopes, so that it has at least two elements; if c is a $(d-1)$-chain different from the "full" $(d-1)$-chain and the empty chain, then it is not in the range of ∂_d, since by stipulation *all* $(d-1)$-polytopes are incident to the unique d-polytope p. The proof is complete.

3 Overview of the Formalization

In this section we describe a formalization of Poincaré's proof of Euler's polyhedron formula that was carried out in the MIZAR system. MIZAR is based on classical first-order logic with equality and Tarski-Grothendieck set theory, a strong theory of sets that is equivalent to the Zermelo-Fraenkel theory together with an axiom asserting the existence of an inaccessible cardinal.

Among the many candidate systems (*e.g.*, ISABELLE, HOL LIGHT, COQ) with which the formalization could have been carried out, MIZAR was selected because of its familiar logical foundations (first-order set theory), its everyday knowledge representation language (dependent types, structures, flexible notation for functions and predicates), its standard proof language (a kind of natural deduction), and its large library of formalized mathematical knowledge on which one can build.[8] But it must be admitted that the choice of MIZAR over the other candidates was somewhat arbitrary. Nonetheless, it seems plausible that, if one were to compare the formalization in MIZAR under discussion with a formalization of the same proof in some other system, one would find considerable overlap.[9]

3.1 Main Formalizations

One often finds when formalizing that, in addition to the logical and mathematical details in a formal proof that must be supplied, one must also formalize various kinds of "background" knowledge. And one often finds that the simplest mathematical facts are (apparently) missing from the library of formalized mathematics[10]. Like Euler writing to Goldbach, we can be surprised that "these general results have not been previously noted by anyone".[11] The formalization of Poincaré's proof of Euler's polyhedron formula in MIZAR was no exception to this phenomenon. But this is understandable; just as libraries of implemented algorithms for various programming languages do not eliminate the need for programmers to adjust them to their specific problems, so too do general mathematical facts in a formal library require further specification before they can be applied.

The contribution naturally divided into three MIZAR "articles" (collections of definitions, theorems). They were:

[8] At the time the formalization began, no formal proof of Euler's formula was known. But independently, another formal proof has been carried out in the COQ system[6].

[9] It would be interesting to discover cases where one *learns* something different about a proof (and not about the different systems or the different logics on which they are built) when formalizing it in one system as compared with what one learns from another formalization of the same proof.

[10] There are two kinds of missing knowledge: well-known (perhaps named) mathematical results can be contrasted with details that, in an less formal context, are left tacit.

[11] And, conversely, often one discovers that mathematical knowledge that we previously thought to be unformalized does in fact exist in the library. At one point the author thought that he had a *proof* that the MIZAR library did not contain a formalization of the fact that $\{0, 1\}$ can be made into a two-element field. This turned out to be mistaken.

- `RANKNULL`: The rank+nullity theorem;
- `BSPACE`: The vector space of subsets of a set based on disjoint union; and
- `POLYFORM`: Euler's polyhedron formula.

We now briefly discuss some notable features of these formalizations.

The rank+nullity theorem The rank+nullity theorem states that if T is a linear transformation from a finite-dimensional vector space V to a finite-dimensional space W, then

$$\dim V = \dim \operatorname{im} T + \dim \ker T.$$

We were able to straightforwardly formalize a standard proof [19] of the result, but some formal groundwork had to be laid for that to be possible.

Much basic linear algebra has already been formalized in MIZAR; there are a number of theorems and definitions concerning subspaces [30], linear combinations [29], dimensions of vector spaces [33] and linear spans of sets of vectors [28]. But some of the linear algebraic facts involved in a proof of the rank+nullity theorem were unavailable and had to be formalized. To carry out the formalization, we defined:

1. the image and kernel of a linear transformation, and the fact that these form subspaces of the domain and range of a linear transformation;
2. the restriction of a linear combination to a set of vectors; and
3. the image and inverse image of a linear combination under a linear transformation.

The first item is straightforward, but the second and third items may require some explanation. In MIZAR, a linear combination is represented as a function from a vector space to the field of scalars whose carrier (the set of vectors not mapped to zero) is finite.[12] The restriction of a linear combination l on a vector space V to a subset X of V is thus naturally represented by the function

$$\lambda v \in V. \begin{cases} l(v) \text{ if } v \in X \\ 0_V \quad \text{otherwise} \end{cases}.$$

Suppose that T is a linear transformation from a vector space V to a vector space W, both over a field F, and that l is a linear combination of vectors in V. Thus l represents the linear combination

$$a_1 v_1 + \cdots + a_n v_n,$$

where n is a natural number, $a_k \in F$ and $v_k \in V$ and $a_k \neq 0_F$ $(1 \leq k \leq n)$. Since T is a linear transformation, we ought to have

$$T(a_1 v_1 + \cdots + a_n v_n) = a_1 T(v_1) + \cdots + a_n T(v_n).$$

[12] This is a case where a representation of a mathematical object contains more information than meets the eye. When represented this way, linear combinations tacitly build in the commutativity of vector addition. $u+v$ is represented by a function f that sends u and v to 1 and every other vector to 0. The same function f also represents $v+u$.

Thus, it is natural to define the image of l under T to be the MIZAR-linear combination

$$\lambda w \in W. \begin{cases} l(T^{-1}(\{w\})) & \text{if } w \in \operatorname{im} T \\ 0_F & \text{otherwise} \end{cases}.$$

The problem with this definition is that it works only if T is injective. We are supposed to define the image of any linear transformation T on any linear combination l, so we need to allow for the possibility that some of the $T(v_i)$'s are equal. A definition that gets around this problem is

$$T(l) := \lambda w \in W. \sum l(T^{-1}(w)).$$

This definition allows us to add together the coefficients, given by l, of those vectors in V that are identified by T. It is interesting to note how the formal definition of the image of a linear combination under a linear transformation differs from the informal (or semi-formal) notation above. This case provides an interesting example of a formal analysis of informal notation.

The inverse image operation also deserves to be mentioned. Suppose that X is a subset of a vector space V, that T is a linear transformation from V to W, and that l is a linear combination of $T(X)$ (that is, that l is a function from W to F with finite support whose value is 0_F outside of $T(X)$). This is a precise way of saying that l looks like

$$b_1 T(v_1) + \cdots + b_n T(v_n),$$

for some natural number n and $v_k \in X$. We want to say that the inverse image of l is the linear combination

$$b_1 v_1 + \cdots + b_n v_n.$$

This is correct, but only on the assumption that the vectors $T(v_1), \ldots, T(v_n)$ are distinct. One way to ensure this is by requiring that $T|X$ is one-to-one, and that is in fact what we did when defining the inverse image operation in MIZAR and suited the formalization task at hand. As it stands, the inverse image operation in MIZAR is a partial operation. The restriction of injectivity of the restriction is, however, unnecessary and it would be valuable to extend the formalization to account for the general case.

The vector space of subsets of a set based on disjoint union Another result needed for a formalization of Poincaré's proof of Euler's polyhedron formula is the fact that the power set of a set forms a vector space over the two-element field F_2. Vector addition is disjoint union (symmetric difference), and scalar multiplication is defined by

$$0 \cdot x := \emptyset, 1 \cdot x := x.$$

This fact seems to be standard, but we were unable to find any conventional name for this space. For lack of a better notation, let $B(X)$ (for "Boole") be the vector space of subsets of X based on disjoint union.

Approximately half of the article `BSPACE` is devoted to proving that $B(X)$ is indeed a vector space. The other half is devoted to some facts about the linear algebraic features of the singleton subsets of X, namely that

– they are a linearly independent set of vectors, and
– if X is finite, then they span $B(X)$.[13]

Polyhedra Perhaps surprisingly, the formalization of Poincaré's proof was rather straightforward. The highlight of the article is the generalized relation, as well as special cases for one-, two-, and three-dimensional polyhedra. The statement of the main theorem, in the MIZAR syntax, is

```
p is simply-connected implies p is eulerian;
```

where of course `p` has type `polyhedron`. The term "Eulerian" is a neologism that means that a polyhedron satisfies Euler's relation; it appears in Lakatos [18]. The definitions of the two properties are

```
p is simply-connected
 means
for k being Integer
 holds k-circuits(p) = k-bounding-chains(p);
```

and

```
p is eulerian
 means
Sum (alternating-proper-f-vector(p))
  = 1 + (-1)|^(dim(p)+1);
```

(The f-vector of a polyhedron p is the sequence

$$s := N_{-1}, N_0, N_1, \ldots, N_d,$$

where $d = \dim p$ and N_k is the number of polytopes of dimension k. It could also be reasonably defined as a bi-infinite sequence indexed by the integers containing the terms displayed above with all other terms being 0. The terminology is standard [4], but to ease the formalization two related neologisms were coined: *proper f-vector* and *alternating proper f-vector*. By definition deleting the first and last terms of s gives the proper f-vector of p; alternating the signs of the sequence yields the alternating proper f-vector of p.) We also proved a lemma on telescoping sums that apparently did not exist in the MIZAR library:

```
for a,b,s being FinSequence of INT
 st len s > 0 &
    len a = len s & len b = len s &
    (for n being Nat st 1 <= n & n <= len s
      holds s.n = a.n + b.n) &
    (for k being Nat st 1 <= k & k < len s
      holds b.k = -(a.(k+1)))
 holds Sum s = (a.1) + (b.(len s))
```

[13] The condition of finiteness is necessary because linear combinations must be finite; if X is infinite no finite linear combination of singletons can equal X.

The lemma is a formalization of the claim that if s, a, and b are are sequences of integers, all of the same length n, and if $s = a + b$ but $b_k = -a_{k+1}$, then $\sum s = a_1 + b_n$. In Poincaré's proof, thanks to the assumption of simple connectedness, the sum on the left-hand side of the Euler relation turns out to be telescoping in this way.

4 Discussion

4.1 Filled gaps

One of the aims of formal mathematics is to give gap-free proofs of mathematical theorems. One could take a skeptical view and doubt the validity of virtually every proof in mathematics; for the skeptic, all proofs are informal and are (potentially) rife with logical gaps. There is a kernel of truth in the skeptical view, but the paucity of *interesting* gaps—oversights, ambiguities, or errors that, once exposed, would alter the views of the working mathematician—makes the view less plausible [9, 25]. One might say that a formalized proof of a theorem gives us better grounds to believe the theorem than were available before the proof was formalized, but at present it seems to be an open philosophical challenge to say why this should be so, while acknowledging the rarity of interesting gaps.

Were any interesting gaps uncovered in the formalization of Poincaré's proof of Euler's polyhedron formula? If a gap can be both interesting and small, then the answer might be "yes", but is more likely "no". In Coxeter's *Regular Polytopes*, we apparently see a proof that "the boundary of any $(k + 1)$-chain is a k-circuit" [5]. But this simply cannot be proved because there are counterexamples.

But it is not clear whether Coxeter is making an invalid inference here. An alternative explanation is that, rather than *proving* that $\partial\partial \equiv 0$ for all polyhedra, Coxeter was instead *motivating* the assumption of this property. Lakatos seems to have observed as much; in his discussion of Poincaré's proof, we find this exchange:

> Gamma: *I think that the boundary of a decent k-chain should be closed. For instance I could not possibly accept as a polyhedron a cube with the top missing; and I could not possibly accept as a polygon a square with an edge missing. Can you prove, that the boundary of any k-chain is closed?*
> Epsilon: *Can I prove that the boundary of the boundary of any k-chain is zero?*
> Gamma: *That is it.*
> Epsilon: *No, I cannot. This is indubitably true. It is an axiom. There is no need to prove it.*

Lakatos is right that this principle (that $B_k \subseteq Z_k$) must be an "axiom" in some form. In the formalization under discussion, it is contained in the definition of simple connectedness.

4.2 A proof-theoretic corollary

The result of the formalization is that Euler's polyhedron formula (understood à la Poincaré) is a first-order logical consequence of the axioms of Tarski-Grothendieck

set theory (TG). But it should be clear that the full strength of TG set is not *required* for Poincaré's proof; it would be quite surprising if Poincaré's proof of Euler's polyhedron formula required the existence of infinitely many inaccessible cardinals. After all, following Poincaré, polyhedra are conceived as certain combinatorial structures that, presumably, could be completely captured in an arithmetical theory. And thanks to the fact that our work on a formal version of Euler's polyhedron formula is quite detailed, one has a clear basis with which to start proving Euler's polyhedron formula in a weaker theory than TG.

The characteristic axiom of TG asserts: for every set N there exists a set M such that

- $N \in M$,
- M is closed under taking subsets,
- M is closed under the powerset operation, and
- if $X \subseteq M$ and $X \not\sim M$, then $X \in M$.

Such a set M might be called a universe containing N; accordingly, let us call this principle the *universe axiom*. Some important consequences of the universe axiom (none of which are axioms of TG) are:

- The existence of an infinite set,
- The axiom of choice, and
- Powerset.

When one inspects the deduction underlying the MIZAR proof of Euler's polyhedron formula, one can trace the argument through each of the three principles mentioned above. Since each of these three principles are consequences of the universe axiom (together, of course, with other axioms of TG), we see that the MIZAR proof of Euler's polyhedron formula uses the universe axiom. But in MIZAR this is to be expected. Indeed, the proof of *every* theorem in the MIZAR mathematical library that involves natural numbers uses the universe axiom by way of the existence of an infinite set (obtained by applying the universe axiom to $\emptyset$).

It may be somewhat surprising that the axiom of choice appears in the proof of Euler's polyhedron formula. To be clear, what is claimed is not that Euler's polyhedron formula ineliminably *depends* on the axiom of choice in the way that, say, the well-ordering principle does. Instead, what is claimed is that there is a deduction of Euler's polyhedron formula that *uses* choice. The use occurs in the proof of the rank+nullity theorem theorem. The proof proceeds by starting with a linear transformation T from a finite-dimensional vector space V to a finite-dimensional vector space W. The first step is to choose a basis A for $\ker T$; one then extends A to a basis B for all of V and, finally, one shows that $T(B - A)$ is a basis of $\operatorname{im} T$. In the actual MIZAR proof of the rank+nullity theorem, the justification for the first step (choosing a basis for $\ker T$) appeals to the theorem [28] that every vector space has a basis.[14]

[14] In the MML version 4.110.1033, released September 9, 2008, the exact MIZAR item is `VECTSP_7:def 3`. Every type in MIZAR must be provably non-empty. Interestingly, the theorem that every vector space has a basis appears not as a MIZAR theorem *per se*, but

But clearly the principle that every vector space has a basis (which, perhaps surprisingly, is equivalent over ZF [3] to the axiom of choice) is stronger than what is required for the purpose of proving the rank+nullity theorem, which after all deals with only finite-dimensional vector spaces.[15] And for finite-dimensional vector spaces, it is clear that we can produce a basis through an iterative search procedure whose formalization requires only arithmetical principles.

Some custom software (building on Josef Urban's work [31]) for computing dependency relations in MIZAR texts provides evidence that the *only* way that the universe axiom is used is by way of the three principles mentioned above (infinity, choice, powerset). This in turn is evidence that, from the provability judgment TG $\vdash$ EPF we have the improved judgment ZFC $\vdash$ EPF, where "EPF" is the Poincaré/combinatorial formalization of Euler's polyhedron formula.[16]

Applying "Kreisel's trick" to the Poincaré/combinatorial understanding of Euler's polyhedron formula, from the judgment ZFC $\vdash$ EPF we can drop choice and conclude that ZF $\vdash$ EPF. We have thus moved from the heights of TG to the more modest realm of ZF by studying the MIZAR deduction of Euler's polyhedron formula; we have established a new provability judgment without actually producing a new deduction.

One can continue the process of trying to further weaken the theory with which proof is carried out. It seems plausible that one can get away without having a *set* of natural numbers. That is, it seems plausible that one can eschew the axiom of infinity and deal with the natural numbers not as a set but as a proper class. Accepting that for the moment, we see, using the equivalence of ZF $-$ Infinity and Peano Arithmetic (PA), that Poincaré's proof of Euler's polyhedron formula can be carried out in PA.

Based on some initial studies, it appears that a formalization of Poincaré's proof can be carried out in the theory $\mathrm{I}\Delta_0(\exp)$, a first-order arithmetical theory in a language with addition, multiplication, ordering, and exponentiation with an induction scheme for Δ_0-formulas (which are permitted to contain exponentiation) [14]. It also appears that some kind of exponentiation is required. These are results in progress and have not yet been rigorously proved.

---

rather as the justification for the non-emptiness of the type `Basis of V`, where V itself has the dependent type `VectSp of F`, where, finally, F has type `Field`. The proof of the non-emptiness of the `Basis` type appeals to the theorem that every linearly independent subset of a vector space can be extended to a linearly independent spanning set, *i.e.*, a basis.

[15] Simpson has shown that the principle "Every vector space has a basis" is equivalent, over the second-order arithmetical theory RCA_0 (for "recursive comprehension axiom"), to the principle of arithmetical comprehension [26].

[16] The custom code is not yet complete; certain features of the MIZAR system are not yet accounted for, such as so-called registrations and the implicit uses of Hilbert's ε-operator. Thus it is possible that some important dependency relations are not being taken into account with the present version of the software.

4.3 Streamlining the formalization

At the time of writing, no mechanism for binders (apart from the quantifiers $\forall$ and $\exists$) has been implemented in the MIZAR language. (Wiedijk has a proposal [32] for this as-yet-unimplemented feature.) For example, the definition of the so-called incidence sequence $I_{x,c}$ generated by a $(k-1)$-polytope x and a k-chain c. Using one common notation for sequences [11], $I_{x,c}$ can be defined as

$$\langle v@P_{k,n} \cdot [x \in P_{k,n}] \colon 1 \leq n \leq N_{p,k} \rangle,$$

The bracket notation "$[x \in P_{k,n}]$", from Knuth [17], denotes 1 or 0 according as the relation does or does not hold.[17] The actual MIZAR definition is somewhat more complicated:

```
incidence-sequence(x,v) -> FinSequence of F2
 means
((k-1)-polytopes(p) is empty implies it = <*>{}) &
((k-1)-polytopes(p) is non empty implies
 len it = num-polytopes(p,k) &
 for n being Nat
  st 1 <= n & n <= num-polytopes(p,k)
  holds
  it.n =
   (v@(n-th-polytope(p,k)))*incidence-value(x,n-th-polytope(p,k)));
```

A binder syntax would simplify this definition. It would also help to simplify the examples involving linear combinations that have already been discussed (in light of the fact that in MIZAR linear combinations are represented as functions). Even if these examples are unconvincing, it should be clear that, in general, notations for sequences, functions (λ-abstraction), relations, and other mathematical objects would help to streamline the MIZAR language and make it even more attractive as a formal language for mathematics than it already is.

5 Further Work

Poincaré's abstract, combinatorial conception of polyhedra facilitated formalization because the definition could be easily captured using MIZAR structures. Following Poincaré, the messy details are largely suppressed; one just formalizes the definition of simple connectedness and carries out the linear algebraic proof. Whether one regards this as a problem or a feature of Poincaré's approach is left for the reader to decide. A further challenge for formal mathematics would be to treat Euler's proof of his relation, involving "concrete" or "real" polyhedra. One could start with the relatively easy case of convex polyhedra (with which Euler was arguably working [10], even though his definition apparently permits non-convex polyhedra). It would be especially interesting to take on Euler's argument because of the subtle

[17] Perhaps even this notation could be implemented in MIZAR, but its logical properties are peculiar and would be a challenge to formally specify.

flaws that it was found to contain. The main problem was that Euler did not specify just how to carry out the slicing procedure. One can see, by inspecting simple examples, that one must be careful about the vertex about which the slicing procedure is done, because for some polyhedra and some choices of the vertex, Euler's method can lead to strange results:

> *It is not at all obvious that this slicing procedure can always be carried out, and it may give rise to 'degenerate' polyhedra for which the meaning of the formula is ambiguous.* [2]

Samelson [23] has repaired this gap in Euler's proof. Are there any others?

As mentioned earlier, for the purposes of the formalization is was not necessary to define in full generality the notion of the inverse $T^{-1}(l)$ of a linear combination l under a linear transformation T. It would be valuable for future formalizations in MIZAR of linear algebra to deal with the full generality of inverse images.

The property of a polyhedron satisfying $\partial\partial \equiv 0$ is part of the definition of simple connectedness. This property is equivalent to the inclusion $B_k \subseteq Z_k$, which says that boundaries are circuits. One might regard this not as the *definition* of simple connectedness, but rather as part of the definition of polyhedron; one would then define simple connectedness as the converse inclusion $Z_k \subseteq B_k$ (circuits are boundaries). For future formalizations using combinatorial polyhedra in MIZAR, it may be valuable (if not necessary) to carry out this rearrangement.

A further step would be to give a formal proof of Steinitz's theorem relating convex "analytic" polyhedra (whose points are in $\mathbf{R}^3$) to planar graphs [12, 20, 4].

References

1. Alama, J.: Euler's polyhedron formula. Formalized Mathematics **16**(1), 2008, 7–17.
2. Biggs, N.L., Lloyd, E.K., Wilson, R.J.: Graph Theory: 1736-1936. Oxford University Press, 1976.
3. Blass, A.: Existence of bases implies the axiom of choice. In Baumgartner, J.E., Martin, D.A., Shelah, S., eds.: Axiomatic Set Theory. Volume 31 of Contemporary Mathematics Series. American Mathematical Society, 1984, 31–33.
4. Brøndsted, A.: An Introduction to Convex Polytopes. Graduate Texts in Mathematics. Springer, 1983.
5. Coxeter, H.S.M.: Regular Polytopes. Dover Publications, 1973.
6. Dufourd, J.F.: Polyhedra genus theorem and Euler formula: A hypermap-formalized intuitionistic proof. Theoretical Computer Science **403**(2-3), August 2008, 133–159.
7. Euler, L.: Demonstratio nonnullarum insignium proprietatum quibus solida hedris planis inclusa sunt praedita. Novi Commentarii Academiae Scientarum Petropolitanae **4**, 1758, 94–108.
8. Euler, L.: Elementa doctrinae solidorum. Novi Commentarii Academiae Scientarum Petropolitanae **4**, 1758, 109–140.
9. Fallis, D.: Intentional gaps in mathematical proofs. Synthese **134**, 2003, 45–69.
10. Francese, C., Richeson, D.: The flaw in Euler's proof of his polyhedral formula. American Mathematical Monthly **114**, April 2007, 286–296.
11. Gries, D., Schneider, F.B.: A Logical Approach to Discrete Math. Springer, 1993.
12. Grünbaum, B.: Convex Polytopes. 2nd edn. Number 221 in Graduate Texts in Mathematics. Springer, 2003.

13. Grünbaum, B.: Polyhedra with hollow faces. NATO-ASI Series C Mathematical and Physical Sciences **440**, 1994, 43–70.
14. Hájek, P., Pudlák, P.: Metamathematics of First-Order Arithmetic. Perspectives in Mathematical Logic. Springer-Verlag, 1993.
15. Hilton, P., Pedersen, J.: Descartes, Euler, Poincaré, Pólya—and polyhedra. L'Enseignement Mathématique (IIe Série) **27**(3-4), 1981, 327–343.
16. Juskevich, A.P., Winter, E., eds.: Leonhard Euler und Christian Goldbach: Briefwechsel 1729–1764. Akademie-Verlag, Berlin, 1965.
17. Knuth, D.E.: Two notes on notation. American Mathematical Monthly **99**(5), May 1992, 403–422.
18. Lakatos, I.: Proofs and Refutations: The Logic of Mathematical Discovery. Cambridge University Press, 1976.
19. Lang, S.: Algebra. Number 211 in Graduate Texts in Mathematics. Springer, 2002.
20. Lindström, B.: On the realization of convex polytopes, Euler's formula and Möbius functions. Aequationes Mathematicae **6**(2-3), June 1971, 235–240.
21. Poincaré, H.: Complément à l'analysis situs. Rendiconti del Circolo Matematico di Palermo **13**, 1899, 285–343.
22. Poincaré, H.: Sur la généralisation d'un théorème d'Euler relatif aux polyèdres. Comptes Rendus de Séances de l'Academie des Sciences **117**, 1893, 144.
23. Samelson, H.: In defense of Euler. L'Enseignement Mathématique **42**, 1996, 377–382.
24. Sandifer, E.: How Euler did it: V, E and F (Part 2). MAA Online, July 2004.
25. Sherry, D.: On mathematical error. Studies in History and Philosophy of Science **28**(3), September 1997, 393–416.
26. Simpson, S.G.: Subsystems of Second Order Arithmetic. Perspectives in Mathematical Logic. Springer, 1999.
27. Spivak, M.: A Comprehensive Introduction to Differential Geometry. 3rd edn. Publish or Perish, 1999 (5 vols.).
28. Trybulec, W.A.: Basis of vector space. Formalized Mathematics **1**(5), 1990, 883–885.
29. Trybulec, W.A.: Linear combinations in a vector space. Formalized Mathematics **1**(5), 1990, 877–882.
30. Trybulec, W.A.: Subspaces and cosets of subspaces in vector space. Formalized Mathematics **1**(5), 1990, 865–870.
31. Urban, J.: XML-izing MIZAR: Making semantic processing and presentation of MML easy. In: MKM 2005: Mathematical Knowledge Management, 2005.
32. Wiedijk, F.: A proposed syntax for binders in MIZAR. Available at `http://www.cs.ru.nl/~freek/mizar/binder.pdf`.
33. Żynel, M.: The Steinitz theorem and the dimension of a vector space. Formalized Mathematics **5**(3), 1996, 423–428.

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 18 (31) 2009

Two Formal Approaches to Rough Sets

Adam Grabowski and Magdalena Jastrzębska

Institute of Mathematics
University of Białystok
ul. Akademicka 2
15-267 Białystok, Poland
adam@math.uwb.edu.pl, magja@osiedle.net.pl

Abstract. The formalization of rough sets in a way understable to machines seems to be far beyond the test phase. For further research, we try to encode the bunch of classical papers within RST and as the testbed of already developed foundations of the theory we try to adopt the interval set model to put it within the existing set-theory machinery in the Mizar computer-checked repository.

1 Preliminaries

During the past decades, mathematics evolved from the pen-and-paper model in the direction of extensive use of computers. Digitization of mathematical journals gets more and more popular, and it is often the case of

- the new material, when papers can be published faster, so information exchange, and hence research is more efficient and accessible – here the well-known example could be Springer's Online First;
- archival issues of journals.

Obviously, simple optical character recognition (OCR) is not the only activity in the latter case – at least the bibliography section could be identified to calculate impact factors properly. These activities can however be done independently, unlike the formalization efforts. If we try to reach the research frontier, i.e. to formalize new results, either solid background should be provided, or the discipline should be relatively new.

We try to address some issues concerning the formalization of a fragment of rough set theory using the Mizar language, presenting a report on the current state of the work. RST delivers important tools to discover data from databases, it is now especially valuable taking into account the amount of stored information and the form of the records. The discipline is rather an emerging trend, and however the stress is put on applications, some valuable results are already available.

The main pros of the use of the Mizar system are as follows:

- repetitions are no longer justified;

ISBN 978-83-7431-229-5 ISSN 0860–150X

– possible generalizations, even those computer-driven;
– possibility of the automatic obtaining new results – the area of knowledge discovery.

The paper is organized as follows. The next section is devoted specifically to the Mizar library, one of the leading mechanical repositories in the world, then follows an outline of one of the most popular models for rough sets is given – namely interval sets. Finally we describe our formalization efforts. The final section brings some concluding remarks and the plans for future work.

2 The Repository

Formalization is a term with a broad meaning which denoted rewriting the text in a specific manner, usually in a rigorous (i.e. strictly controlled by certain rules), although sometimes cryptic language. Obviously the notion itself is rather old, originated definitely from pre-computer era, and in the early years formalization was to ensure the correctness of the approach. As the tools evolved, the new paradigm was established: computers can potentially serve as a kind of an oracle to check if the text is correct.

The problem with computer-driven formalization is that it draws the attention of researchers somewhere at the intersection of mathematics and computer science, and if the complexity of the tools is too high, only software engineers will be attracted and all the usefulness for an ordinary mathematician will be lost.

The Mizar Mathematical Library (MML for short) established in 1989 is considered one of the bigest repositories of computer checked mathematical knowledge in the world. The basic item in the MML, called a Mizar article, reflects roughly a structure of an ordinary paper, being considered at two main layers – the declarative one, where definitions and theorems are stated and the other one – proofs. Naturally, although the latter is the larger part, the former needs some additional care – the submission will be accepted for inclusion into the MML if the approach is correct and the topic is not already present there.

In recent years, the most intensively developed disciplines were general topology (steered by Trybulec, Białystok, Poland) and functional analysis (led by Shidama, Nagano, Japan). The first author took part in a large project of translating a compendium *Continuous Lattices and Domains* with a significant success. As a by-product of this encoding, apart from quite readable Mizar scripts, also the presentation of the source which is accessible to ordinary mathematicians and pure HTML form with clickable links to notions and theorems are available.

Since mathematics in MML is based on ZFC set theory, and the notion of structures is of somewhat other character, the basic division of the repository into two parts is provided: the articles which do not use the notion of a structure (forming the so called concrete part) and the remaining, abstract part of MML. This division stimulates the enhancement of the library, forcing the movement of preliminary concrete items from the abstract part of the library.

This can be considered a drawback of the MML – obviously the chosen set theory cannot be changed easily, and the formalization of work which strongly

depends (or just proposes another axiomatics) on a set theory other than ZFC, e.g. Bryniarski's work [1] is not that straightforward.

3 The Theory, Informally

There are two popular extensions of the classical set theory – one of them, Zadeh's fuzzy sets, are already present in the MML [7]. The rough-set (and corresponding interval-set) model is another important extension for modelling vagueness, where information is incomplete or imprecise. One of the views for rough sets is operator-oriented, with approximations defined as two additional set-theoretic objects. The other is set-oriented view, taking a rough set as the family of sets having the same upper and lower approximations. In the interval-set model, the range of the unknown set is given as a pair of its bounds. Hence virtually any member of such family can be the considered set. Although both models have strong logical foundations, we will focus here rather on the construction of appropriate algebraic models.

3.1 Rough sets

Rough sets, which were introduced by Pawlak [10], are often viewed through the prism of applications, especially when the information is incomplete or uncertain. Given a finite non-empty universe U and the relation R defined on U which in the original Pawlak's approach is an equivalence relation, i.e. is reflexive, symmetric and transitive, a partition U/R of U into equivalence classes E_i, called elementary sets, is given. The pair $\langle U, R\rangle$ is called an approximation space. From the agent's point of view, elements which belong to the same class of R are indiscernible.

In the approximation space $\langle U, R\rangle$ one can define for an arbitrary set $A \subseteq U$ the two operators of the lower and upper approximation.

$$A_\star = \bigcup_{E_i \subseteq A} E_i = \{x \in U : [x]_R \subseteq A\}$$

$$A^\star = \bigcup_{E_i \cap A \neq \emptyset} E_i = \{x \in U : [x]_R \cap A \neq \emptyset\}$$

Potentially, we can consider elementary sets as just elements of a partition of U with R as a hidden argument. It is clear that if equivalence classes are singletons, then the rough set, treated as a pair of $A_\star$ and $A^\star$ reduces to the ordinary set A. Lattices of rough sets were studied primarily by Iwiński in [5] in the original Pawlak's setting. In our case we drop the assumption of reflexivity of R, i.e. in case of symmetric and transitive relations, or in case of tolerance relations, the lattice retains some basic properties. This generalization can go even further, with the obvious change from $[x]_R$ into the image $R(x)$.

3.2 Interval sets

According to the interval-set approach, over a finite non-empty universe U we can think of the interval set $\mathcal{A}$ as a pair of two sets, $A_1, A_2 \in 2^U$, where $A_1 \subseteq A_2$ as follows:

$$\mathcal{A} = [A_1, A_2] = \{X \in 2^U : A_1 \subseteq X \subseteq A_2\}.$$

Given two intervals, $\mathcal{A} = [A_1, A_2]$ and $\mathcal{B} = [B_1, B_2]$. We define the interval-set intersection, union and difference as

$$\mathcal{A} \sqcap \mathcal{B} = \{A \cap B : A \in \mathcal{A}, B \in \mathcal{B}\}$$

$$\mathcal{A} \sqcup \mathcal{B} = \{A \cup B : A \in \mathcal{A}, B \in \mathcal{B}\}$$

$$\mathcal{A} \setminus \mathcal{B} = \{A \setminus B : A \in \mathcal{A}, B \in \mathcal{B}\}$$

Further on, we can define the interval-set complement by putting

$$\neg[A_1, A_2] = [U, U] \setminus [A_1, A_2].$$

It is clear that $I(2^U)$, the set of all intervals over U, together with the above operations, forms a completely distributive lattice which is not a Boolean algebra, since for an interval set A, $A \sqcap \neg A$ is not necessarily equal to $[\emptyset, \emptyset]$, similarly $A \sqcup \neg A$ is not necessarily equal to $[U, U]$.

4 Formalization of Intervals

In this section we describe briefly main points of the formalization of (lattices of) intervals we recently completed in [4].

Tolerance approximation spaces, i.e. the framework in which rough sets are defined, are relational structures. An important fact here is that the formalization of interval sets does not depend on the notion of a structure (essentially the operations on such sets can be viewed as the operations applied to ordered pairs, or as collective operations on families of subsets).

4.1 A mixture of modes and functors

Since appropriate correctness conditions should be proven for all definitions (the existence for both modes and functors and additionally the uniqueness for functors), the Library Committee suggests to use the functors instead of modes when possible to prove that they are uniquely determined (one of the exceptions is given in Section 5.1). According to this policy, we are forced to use the notion of the interval set parametrized by its ends.

```
definition let U be set;
           let X, Y be Subset of U;
  func Inter (X,Y) -> Subset-Family of U equals
:: INTERVA1:def 1
    { A where A is Subset of U : X c= A & A c= Y };
end;
```

But we needed also more general notion, the class of all intervals over given universe.

```
definition let U be set;
  mode IntervalSet of U -> Subset-Family of U means
:: INTERVA1:def 2
    ex A, B be Subset of U st it = Inter (A, B);
end;
```

The last redefinition is to restrict the result type of the functor `Inter` and to ensure that we can use operations defined on itervals to concrete intervals determined by its ends.

```
definition let U; let A, B be Subset of U;
  redefine func Inter (A, B) -> IntervalSet of U;
end;
```

4.2 Operations on intervals

On the one hand, treating interval sets as families of subsets, we can consider lattice operations on them as corresponding set-theoretical collective operations.

```
definition let U be non empty set,
               A, B be non empty IntervalSet of U;
  func A _/\_ B -> IntervalSet of U equals
:: INTERVA1:def 3
    INTERSECTION (A, B);
end;
```

Using the "equals" construction both constructors (i.e. INTERSECTION and "`_/\_`" operation) are automatically unified.

```
definition let SFX, SFY be set;
  func INTERSECTION (SFX,SFY) means
:: SETFAM_1:def 5
     Z in it iff ex X,Y st X in SFX & Y in SFY & Z = X /\ Y;
end;
```

Note that these operations, named INTERSECTION and UNION in the Mizar formalism, in fact were originally defined on families of subsets, and later the types of arguments were generalized just into sets. Of course, basic translation lemmas are provided.

```
theorem :: INTERVA1:12
  for U being non empty set,
      A1, A2, B1, B2 being Subset of U st
        A1 c= A2 & B1 c= B2 holds
    INTERSECTION (Inter (A1,A2), Inter (B1,B2)) =
      { C where C is Subset of U : A1 /\ B1 c= C & C c= A2 /\ B2 };
```

We proved some necessary lemmas – properties of intervals, and instead of using concrete functors (`Inter`), we introduced an attribute with a similar meaning.

```
definition let X be set; let F be Subset-Family of X;
  attr F is ordered means
:: INTERVA1:def 8
    ex A, B being set st
      A in F & B in F & for Y being set holds
        Y in F iff A c= Y & Y c= B;
end;
```

We were surprised why the following was not proven before – Mizar article SETFAM_1 [9] is dated back to 1989; apparently for twenty years nobody had needed this elementary fact of distributivity or just the proof is hidden too deeply (simple searching for simultaneous use of both functors was unsuccessful).

```
theorem :: INTERVA1:30
  for X being set, A,B,C being non empty ordered Subset-Family of X holds
    UNION (A, INTERSECTION (B,C)) =
      INTERSECTION (UNION (A,B), UNION (A,C));
```

4.3 Linking the classical set theory

Note that min and max synonyms were introduced to set-theoretical meet and union, respectively, because interval's ends can be calculated so.

```
theorem :: INTERVA1:27
  for A,B being Subset of U,
      F being ordered non empty Subset-Family of U st
      F = Inter (A,B) holds
      min F = A & max F = B;
```

Speaking informally about the extension of the classical set theory we can avoid some real formal difficulties (as in the case of the set of complex numbers which is an extension of the set of all reals, but was originally defined in the MML as the Cartesian square $\mathbb{R}^2$). Hence, trivial intervals determined by A correspond to the singletons of A.

4.4 The complementation operator

The complementation has a slightly different notation than the ordinary set-theoretic one in order to avoid the overloading (remember A is a family of subsets of the universe U, and hence a subset of 2^U):

```
definition let U be non empty set, A be non empty IntervalSet of U;
  func A ^ -> non empty IntervalSet of U equals
:: INTERVA1:def 10
    Inter ([#]U,[#]U) _\_ A;
end;
```

Similar overloading would be dangerous in the case of projections `‘1` and `‘2` – originally these were just the coordinates of the Cartesian product. We defined the `A‘‘1` and `A‘‘1` as the left-hand-side and the right-hand-side ends of the interval A (or the meet and the union of A).

```
theorem :: INTERVA1:46
  for U being non empty set, A being non empty IntervalSet of U holds
    A^ = Inter ((A‘‘2)‘,(A‘‘1)‘);
```

Within the Mizar library some useful special objects are constructed when needed to register some existential clusters, and counterexamples are considered usually to illustrate the topic to students. Since interval sets act in some cases not as classical crisp sets, we find it useful to also include them in the article [4].

```
theorem :: INTERVA1:54
  for A being non empty IntervalSet of U holds
    {} in A _/\_ (A^);
```

5 Rough Sets Revisited

The problem with the rough sets is that in order to fully reuse the expressive power of the Mizar language and to reflect the current state of the MML, unlike the intervals, rough sets should be defined using the abstract part of the Mizar library. On the other hand, to benefit from the generalized approach to sets via rough sets, it would be necessary to have them in the concrete part. This, however, is impossible, because the rough sets were defined in Mizar over the tolerance space, i.e. the structure belonging to the latter part of MML.

5.1 Pairs vs. subsets

We will not describe here the basic notions of the formalized approximations, the details can be found in [2] and [7]. They are defined in Mizar pretty closely to the natural language according to Section 3.1. The only doubt was whether to choose the definition of a rough set as a union of elementary sets or the latter one, in terms of equivalence classes. The former can omit the notion of indiscernibility relation, but we decided to follow the latter way as it is used more often. Obviously, none of the properties of indiscernibility relation, neither transitivity, nor even symmetry is assumed. The properties are added later to show essential properties of the operators, only where needed.

For example, the lower approximation $X_\star$ of a rough set X in the approximation space A is given by

```
definition let A be non empty RelStr;
           let X be Subset of A;
  func LAp X -> Subset of A equals
:: ROUGHS_1:def 4
  { x where x is Element of A : Class (the InternalRel of A, x) c= X };
end;
```

where `Class` is just another name for the image of a relation (as a result of a revision, originally the class of abstraction of an equivalence relation).

```
definition
  let A be Approximation_Space;
  let X be Subset of A;
  mode RoughSet of X means
:: ROUGHS_1:def 8
    it = [LAp X, UAp X];
end;
```

Potentially, this mode can be defined as a functor (it is unique), but we found it useful to have both views for rough sets formalized verbatim (with the other representation just as the subset of the tolerance approximation space, being rough in the case of the different lower and upper approximations, and exact or crisp, otherwise).

```
definition let X be Tolerance_Space;
  mode RoughSet of X ->
    Element of [:bool the carrier of X, bool the carrier of X:] means
:: INTERVA1:def 13
    not contradiction;
end;
```

As it can be easily seen, both notions coincide, but to ensure the correspondence between the two models, the following operator which converts a subset of approximation space (with the tolerance relation as a hidden argument) into the pair of sets was introduced:

```
definition let X be Tolerance_Space, A be Subset of X;
  func RS A -> RoughSet of X equals
:: INTERVA1:def 14
    [LAp A, UAp A];
end;
```

5.2 Lattice-theoretical approach

In many real-life applications, lattice theory usually serves well as the source of appropriate models, hence many examples of lattices can be found in the MML, e.g. lattices of subgroups, subspaces of a vector space, real numbers or topological domains. Also lattices of fuzzy sets are formalized there.

```
definition let X be Tolerance_Space;
  func RSLattice X -> strict LattStr means
:: INTERVA1:def 23
    the carrier of it = RoughSets X &
    for A, B being Element of RoughSets X,
        A', B' being RoughSet of X st A = A' & B = B' holds
     (the L_join of it).(A,B) = A' _\/_ B' &
     (the L_meet of it).(A,B) = A' _/\_ B';
end;
```

We have shown basic properties of the lattice of rough sets, such as distributivity, boundedness and completeness (Stone algebras are not yet defined in the MML), most of them formulated as functorial registrations of adjective clusters of the form

```
registration let X be Tolerance_Space;
  cluster RSLattice X -> complete;
end;
```

Besides the obvious pros of using a computer math-assistant we can point out here two main gains of this formalization. From the viewpoint of rough-set theory, it is the inheritance – in Mizar the structures can be freely extended, e.g. to apply to the lattice structures given a topology, or an ordering relation (since lattices and posets are developed in some sense independently, or in parallel).

From the viewpoint of the Mizar community the gain is that some evident gaps were identified and filled in, like the aforementioned distributivity of collective intersection and union.

6 Conclusions and Further Work

Regardless of the concrete formal definition of rough sets chosen (either as a pair of approximations or as the equivalence class of indiscernibility relation), many interesting algebraic models of rough sets are presented, see e.g. Pomykała [11] or the aforementioned Iwiński [5]. Also Bryniarski [1] offers a formal approach which is pretty close to the Mizar formalism in its style, although the motivation is somewhat different (and also computers were not used there).

Based on the MML, we can be sure that a thorough exploration of the theory, like the formalization of "Continuous Lattices and Domains" or general topology points out much better existing gaps and possible improvements of this repository. Unlike RST, these disciplines have standard textbooks, and this can make this work harder. Our goal is to formalize (or, more precisely, to map, because many of the facts are already available in the MML) Järvinen's paper [6]. Having constructed the structure of intervals and rough sets, further research will be continued. As a by-product, we can also reuse general topology which is the area where roughness could also be studied – with the obvious example of the upper and lower approximation operators as the topological closure and interior.

In parallel, the authors from China (not directly involved in this project) wrote an article about the properties of rough subgroups, already accepted for inclusion into the MML, as an extension of the existing group theory corpus.

Strenghtening of the Mizar checker will hopefully decrease the de Bruijn factor (the quotient of the size of a formalization of a mathematical text and the size of its informal original), and hence, the proofs will be more compact, with the readability unaffected.

Although existing provers are best known in the area of finding short axiomatizations of various logical systems (like the classical problem of Robbins algebras), other possibilities can enhance this framework. Urban's [13] tools translating the

Mizar language into the input of first-order theorem provers or XML interface providing information exchange between various math-assistants are already in use. Here we can foresee exploration of the properties of certain lattices.

References

1. Bryniarski, E.: *Formal conception of rough sets,* Fundamenta Informaticae 27(2–3), pp. 109–136, 1996.
2. Grabowski, A.: *Basic properties of rough sets and rough membership function,* Formalized Mathematics, 12(1), pp. 21–28, 2004.
3. Grabowski, A.: *On the computer-assisted reasoning about rough sets,* in B. Kęplicz et al. (Eds.), Monitoring, Security and Rescue Techniques in Multiagent Systems, *Advances in Soft Computing,* Springer, pp. 215–226, 2005.
4. Grabowski, A. and Jastrzębska, M.: *On the lattice of intervals and rough sets*, to appear in Formalized Mathematics, 2009.
5. Iwiński, T.: *Algebraic approach to rough sets,* Bulletin of the Polish Academy of Sciences. Mathematics, 35, pp. 673–683, 1987.
6. Järvinen, J.: *Lattice theory for rough sets,* in J.F. Peters et al. (Eds.), Transactions on Rough Sets VI, LNCS 4374, pp. 400–498, 2007.
7. Mitsuishi, T., Endou, N. and Shidama, Y.: *The concept of fuzzy set and membership function and basic properties of fuzzy set operation,* Formalized Mathematics, 9(2), pp. 351–356, 2001.
8. Mizar Home Page, `http://mizar.org/`.
9. Padlewska, B.: *Families of sets*, Formalized Mathematics, 1(1), pp. 147–152, 1990.
10. Pawlak, Z.: *Rough sets,* International Journal of Computer and Information Sciences, 11, pp. 341–356, 1982.
11. Pomykała, J. and Pomykała, J.A.: *The Stone algebra of rough sets,* Bulletin of the Polish Academy of Sciences. Mathematics, 36(7–8), pp. 495–508, 1988.
12. Trybulec, A.: Some Features of the Mizar Language. In Proceedings of ESPRIT Workshop, Torino, 1993, available at `http://mizar.uwb.edu.pl/project/trybulec93.ps`.
13. Urban, J.: *Translating Mizar for first order theorem provers,* LNCS 2594, pp. 203–215, 2003.
14. Yao, Y.Y. and Li, X.: *Comparison of rough-set and interval-set models for uncertain reasoning,* Fundamenta Informaticae, 27(2–3), pp. 289–298, 1996.

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 18 (31) 2009

Improving Representation of Knowledge within the Mizar Library*

Adam Grabowski and Christoph Schwarzweller

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

Abstract. Efficient handling of extensive repositories is one of the major objectives of mathematical knowledge management. It is naturally connected, in order to attract more potential users (both researchers and students), with the need to build large libraries of formalized mathematics, and these two activities should not interfere. This may be achieved by constant enhancement of the quality of digital repositories, in which the proofs can additionally be verified with the help of proof assistants. Based on our experience with the MML we discuss some of the issues concerned with this process, describe mechanisms of revisions which seem to be indispensable to meet the expectations of contemporary mathematicians. We argue that even careful reviewing of contributions cannot cope with the task of keeping a mathematical repository efficient and clearly arranged in the long term.

1 Introduction

The contemporary way of doing mathematics can be substantially enriched by using computers, in comparison to classical pen-and-paper model – and this is what Mathematical Knowledge Management network aims at.

For a researcher, one of the activities here is obviously just doing mathematics for himself – and math assistants known in this areas offer much freedom and flexibility. Certain doubts can appear if it comes to knowledge exchange – incompatible definitions, different notations or more than one proof of the same theorem are something natural in mathematical practice. For repositories of computer-checked knowledge this means that either these various objects should be linked somehow or they should be handled separately, however automatic discovery of such places seems to be hard.

One possibility, of course, is reviewing the submissions. Reviewing improves the quality of knowledge and proofs added to the repository, but we shall illustrate that in the long run it cannot ensure that a mathematical repository meets the expectations. We therefore claim that revisions (as the reorganizations of the Mizar library

* This is an extended and updated version of the paper from MKM 2007.

ISBN 978-83-7431-229-5 ISSN 0860–150X

are usually called) are an essential part of maintaining mathematical repositories: in order to keep it clean and attractive for users, from time to time a "core team" has to check and improve the organization, quality, and proofs of a mathematical repository.

In the following section we describe and discuss the goals and benefits of revisions compared to a straightforward reviewing process. Then, after a brief introduction to the Mizar system [11], we consider the reviewing process of MML submissions in Section 3. We describe reviewing criteria and show which insufficiencies can be handled by reviewing. In contrast, Section 4, is devoted to revisions of MML illustrating on the one hand what kind of improvements reviewing cannot perform, on the other hand the role of revisions in maintaining MML. This is the process done mainly by a human hand as of now, but we also describe existing Mizar utilities devised to facilitate this process; in the next section we discuss some issues concerned with this activity and describe some traps the developers may meet when enhancing the library. Then we conclude drawing some remarks for future.

2 The Motivation

The goal of a revision is to improve the mathematical repository. In contrast to reviewing submissions, however, here the attention is turned to the repository as a whole, not to a single, new part of it. Consequently, the motivation for revisions can be for example:

- keeping the repository as small as possible,
- preserving a clear organization of the repository in order to attract authors,
- establishing "elegant" mathematics, that is e.g. using short definitions (without unnecessary properties) or better proofs.

Note that all these points characterize a qualitative repository and can hardly be achieved by reviewing single submissions. Of course there are different possibilities to achieve the points mentioned. Improving the prover e.g. can shorten proofs and hence – simplify the repository. Reorganizing a mathematical repository probably demands manipulating the whole file structure, not only the files themselves. Therefore we decided to classify revisions based on their occasion, that is on which kind of insufficiency we want to address. Based on our experiences with the Mizar Mathematical Library we distinguish four major occasions for revisions:

1. improving authors' contributions;
2. improving the underlying prover or proof checker;
3. reorganizing the repository;
4. changing the representation of knowledge.

Improving an authors' contributions is the typical task of reviewing and is of course to be recommended for mathematical repositories too: nomenclature can be polished up to fit to the yet existing one, definitions can be improved, that is

e.g. generalized if appropriate. Proofs are also a matter of interest here, especially keeping them as short as possible, yet still understandable is of major concern. In a large, open repository however, authors sometimes may prove and submit theorems or lemmas not being aware that those are already part of the repository. Similarly, special versions of already included theorems can happen to be "resubmitted". It is doubtful that this kind of flaws will be detected by ordinary reviewing.

Strengthening the underlying prover or proof checker has also an impact on the repository. Proofs can be shortened or rewritten in a more clear fashion, both being fundamental properties of attractive mathematical repositories. Even more, theorems in such a collection may become superfluous, because the improved prover accepts and applies them automatically. A typical example here is the additional inclusion of decision procedures.

Reorganizing the repository deals with the fact that a repository is built up by a large number of contributors. For their new developments, authors (should) use already existing theories as a basis. To establish their main results, however, they often have to prove additional theorems or lemmas just because the theory used does not provide them yet. So, these additional facts have to be put in the right place of the repository. Otherwise, it will be hard for other authors to detect them or at least searching the repository becomes less comfortable. The building of monographs goes in the same direction: a frequently used theory should be handled with extra care. Not only should all related theorems be collected in a distinguished place, but also still lacking theorems be complemented, in order to ease the work for further authors. These tasks can only be accomplished when considering the repository as a whole, that is by revisions.

The last point concerns the development of a repository in the long term. What if after a while it turns out that another definition or representation of mathematical objects would serve our purposes better than the one chosen? Should it be changed? Note that a lot of authors already could have used these objects in their proofs, that is changing the definition or representation would imply changing all these proofs – and of course one cannot force authors to redo all their proofs. On the other hand, including both definitions or representations leads to an unbalance: the theory of the new preferred version is much less developed than the one of the old version, so authors hardly will base their developments on the new one. Again, the solution is a revision: in the best case definitions and representations are changed by the "core team", so that ordinary users can further use all theorems without even noticing they have been changed.

In the following sections we will illustrate these considerations by examples taken from the Mizar Mathematical Library and in particular show how revisions maintain mathematical repositories.

3 Reviewing MML Submissions

Reviews of submissions to the MML – as reviews of ordinary submissions for conferences or journals – have the overall goal to check whether a submission should be accepted (for inclusion into the MML) and simultaneously improve the quality of

that submission. For mathematical repositories, however, the criteria for acceptance and improvements are somewhat different.

3.1 The Criteria

Certainly the contents of a submission for a repository should likewise be original and interesting. Original here, of course, means that definitions and theorems presented are not part of the repository, yet. This is easy to check for the main theorem of a submission. For technical lemmas used to establish this main result, however, this task is much more difficult. So, for example, a reviewer will probably neither know nor be willing to check whether a theorem like

```
theorem
  for F,G being FinSequence, k being Nat st
    G = F|(Seg k) & len F = k + 1 holds F = G ^ <*F/.(k+1)*>;
```

is already included in a repository. Even if the textual search via grepping is no longer the only method to find such repetitions since the MML Query by Grzegorz Bancerek [3] is available, even after the volunteer will learn how to use this system, still there is no single automated bunch of tools which removes all repeated theorems effectively. Furthermore, the motivation to check these things in detail will be even decreased, because such a point will not decide between acceptance and rejection.

The question whether a submission is interesting should be handled more liberally. Of course, the usual issues, that is the quality of the main results, apply here too. There is, however, another kind of submissions to repositories: the one that deals with the further development of (basic) theories. This concerns collections of basically simple theorems providing necessary foundations, so that more ambitious developments can be accomplished easier. Usually, these are theorems that easily follow from the definitions, however are used so frequently that repeating the proof over and over again is hardly acceptable. Examples here are the theories of complex numbers or polynomials, where among other things we can find the following theorems.

```
theorem
  for a,b,c,d being complex number st
    a + b = c - d holds a + d = c - b;

theorem
  for n being Ordinal,
      L being add-associative right_complementable add-left-cancelable
          right_zeroed left-distributive (non empty doubleLoopStr),
      p being Series of n,L holds
   0_(n,L) *' p = 0_(n,L);
```

Though hardly interesting from a mathematical point of view, such theorems are important for the development of a repository and should therefore be considered as interesting, too.

Improvements of a submission are a more difficult issue. Firstly, we can consider definitions and notations contained in the submission. Can they be arranged more sparsely, i.e. can the results be established based on fewer axioms? Is it possible or reasonable (in the actual repository) to generalize the definitions? This also applies to theorems. Note that the above theorem, though applicable to polynomials, is in fact stated for power series. Again to address these issues, a good knowledge of the repository by the reviewers is necessary.

Secondly, when it comes to proofs, there are hardly any guidelines, because proving in particular is a matter of style. We can hardly force an author to change his (finished) proof into another one using completely different proof techniques. What we can do, is to suggest improvements for the presented proof. We can, for example, propose a more accurate use of the proof language to get more elegant or better readable proofs. Or we can give pointers to other theorems in the repository that allow to shorten the proof.

3.2 The Evaluation

Based on these considerations the reviewing process for Mizar articles, that is for submissions to the Mizar Mathematical Library, has been introduced. Using basically the commonly used scheme accept/revise/reject (and apart from its descriptive grade) the rating of a submission can be[3]

A. accept
   requires editorial changes only, which can be done by the editors
B. accept
   requires changes by the author to be approved by the editors
C. revise
   substantial author's revisions necessary, resubmission for another review
D. decision delayed
   revision of MML necessary
E. reject
   no hope of getting anything valuable

The most important issue here, of course, is the question whether an article should be included in MML. Note that there are two grades (A and B) for acceptance. The reason is that accepted articles should be included in the MML as soon as possible to avoid duplication of results during the reviewing phase.[4] So, while submissions rated B or C need feedback from the authors, submissions rated A can be added to MML without further delays.

The most interesting point is D. Note that here already the problem of a revision of the whole repository is addressed. Reviewing can point out that – albeit the

[3] There has been and is still going on an email discussion about these options, especially the last grade is questionable since virtually in any paper one can find useful parts. E is kept for articles which cover already formalized topics, blocks of theorems trivial for the checker or just meaningless definitions and their consequences.

[4] There even has been the proposition of making public submissions before reviewing to avoid this problem, but we are not aware of a definite decision concerning this point.

author has proved his main results – the way Mizar and MML support establishing the presented results is not optimal and should be improved.

As the most notable example here, we can cite the newly submitted definition of a kind of a norm for the elements of the real Euclidean plane, which are defined in the Mizar library just as finite sequences of real numbers.

```
definition let n be Nat, f be Element of TOP-REAL n;
  func |. f .| -> Real means
    ex g being FinSequence of REAL st
      g = f & it = |. g .|;
end;
```

where `|. g .|` is a usual Euclidean norm which was introduced in the MML before as

```
definition let f be FinSequence of REAL;
  func |. f .| -> Real equals
:: EUCLID:def 5
    sqrt Sum sqr f;
end;
```

After the change of the loci type (the submission obtained grade D, of course) from `FinSequence of REAL` into `real-yielding FinSequence` in the `EUCLID` article the earlier definition was no longer needed which helped to simplify the structure of notions in this new submission. We can (and eventually did so in December 2007) go even further: it is reasonable to separate all operations on functions fulfilling the same scheme into new library item; that is the way two articles from `VALUED` started – devoted to various functions with numbers as values and their properties.

The possible flattening of the net of notions seems to be highly desirable – another illustrative example can be here the notion of the absolute value, defined originally for real numbers as

```
definition let x be real number;
  func abs (x) -> real number equals
:: ABSVALUE:def 1
    x if 0 <= x
    otherwise -x;
end;
```

and the other, corresponding but fully independent, for complex numbers:

```
definition let z be complex number;
  func |.z.| -> complex number equals
:: COMPLEX1:def 16
   sqrt ((Re z)^2 + (Im z)^2);
end;
```

Because all reals are also complex numbers (it is automatically obtained via the attributes and clusters mechanism), the following modification (such flattening or linearization is called a redefinition) was done:

```
definition let x be real number;
  redefine func |. x .| equals
:: ABSVALUE:def 1
  x if 0 <= x otherwise -x;
end;
```

which needed of course the proof of the equivalence of both notions for reals; `abs` is still kept as a synonym.

3.3 First Impression

The decision is not a typical result of majority voting, because referees giving the C grade point out possible improvements, so usually the lowest grade counts (luckily, in case of E marks, all three referees agreed).

To summarize the grades for 2006, let us look at Table 1.

Table 1. Number of submissions to the MML and their grades in 2006

	all	A	B	C	D	E
items	39	6	4	20	6	3
% of total	100	15.4	10.2	51.3	15.4	7.7

Basically, all ten submissions graded A and B were included into the MML, and among C and D candidate articles, which were returned to authors, other 15 were accepted; in total there were 25 Mizar articles accepted in 2006, the first year when the reviewing procedure as described above was introduced.

All in all we have seen that reviewing MML submissions indeed addresses only the first point mentioned in Section 2. Of course a thorough reviewing process will improve the quality of MML articles and may even guide authors into the direction of a good style of "Mizar writing". As we can conclude from Table 1, this is the case of the majority of submissions because the authors should enhance the articles according to the referees' suggestions. However there remain situations in which the MML as a whole should be improved; in the long term mere reviewing of submissions cannot avoid this. Here even carefully reviewing of Mizar articles – as already indicated by rate D above – can only help to detect the need for such revisions.

3.4 Further Development

In 2007, the policy of reviewing showed its usefulness – all articles with A and B grades were immediately (those with B – after suggested corrections) incorporated into the MML, as well as four among the others. Four articles still wait for a revision of the library. Interesting to see, comparing to the previous year the number of positive grades is granted to nearly 70% submissions. The reasons can be twofold:

Table 2. Number of submissions to the MML and their grades in 2007

	all	A	B	C	D	E
items	50	14	20	10	5	1
% of total	100	28	40	20	10	2

first of all, assuming that majority of authors are active on a long term basis, at least longer than one year, by applying referees' suggestions they worked out their better proof style for their future articles (remembering some of the submissions are revised C-grades from 2006, so the total numbers are bigger). The other reason is that articles can just be written better – thanks to revisions (especially generalizations, removing repetitions, and a better organization of knowledge).

Table 3. Number of submissions to the MML and their grades in 2008

	all	A	B	C	D	E
items	55	8	37	9	0	1
% of total	100	14	68	16	0	2

In the subsequent year, actual D grades were suggested, but they were followed by the detailed description of which changes of the MML should be done, hence appropriate revisions were done immediately. The percentage of A grades is lower; B ones were usually corrected by authors. What is worth noting here is that many C-grades from this year were not improved by the authors and resubmitted, few cases are students who just graduated. The only E was the case of unfortunate repetition of virtually all theorems from an article already accepted to the MML (of course the author did not know this).

4 Improving the Library

The Library Committee has been established on November 11, 1989. Its main aim is to collect Mizar articles and to organize them into a repository – the MML. Recently, from this agenda a new additional one was created – the Development Committee, which takes care of the quality of the library as a whole.

4.1 Types of Revisions

For the reasons we tried to point out before, the Mizar Mathematical Library is continuously revised. Roughly speaking, there are three different kinds of revisions:

– an authored revision – consists of small changes in some articles in the library when somebody writing a new article notices a theorem or a definition in an old

article that can be generalized. This is also the case of D grades as described above. To do this generalization, sometimes it is necessary to change (improve) some older articles that depend on the change. As a rule, a small part of the library is affected, but if it is not the case, usually it is hard to handle more such revisions at a time, otherwise the Library Committee has a lot of work with the synchronization of the patches (some sort of revision control system could be useful here).
- an automatic revision – takes place frequently whenever either a new revision software is developed (e.g. software for checking equivalence of theorems, which enables to remove one or two equivalent theorems) or the Mizar verifier is strengthened and existing revision programs can use it to simplify articles.
- a massive reorganization of the library – although was very rare before, as of now it happens rather frequently. It consists in changing the order of processing articles when the Mizar data base is created. Its main steering force is the division of the MML into the concrete and abstract parts.

4.2 MML Versions

Apart from the Mizar version numbering, the MML has also a separate indexing scheme. As of the time of writing, the latest official distribution of the MML was numbered as 4.128.1063.

As a rule, the last number, currently 1063 shows how many articles there are in the library (this number can be sometimes different because 36 items were removed so far from the MML, but some additional items such as EMM articles, and "Addenda" which do not count as regular submissions, were added). The second number (128) changes if a bigger revision is finished and the version is made official. Although it is relatively small comparing with the age of the library, the changes are much more frequent. To give some numbers, at the beginning of 2006, this value was set to 51, so a revision happens approximately every two-three weeks (or, to be more precise, this is how often it is made public).

4.3 Some Statistics

The policy of the head of Library Committee – to accept virtually all submissions from the developers and, if needed, enhance it by himself, was then very liberal. For these nearly twenty years there were only three persons taking the chair of the head of the committee (Edmund Woronowicz, Czesław Byliński, and currently Adam Grabowski); their decisions were usually consulted with other members of the committee, though.

Such an openness of the repository was justified: in the early years of the Mizar project the policy "to visit Białystok and get acquainted with the system straight from its designers" resulted in the situation that all authors knew each other personally, now the situation changed.

The MML evolved from the project, frankly speaking, considered rather an experiment of how to model mathematics to allow many users benefit from a kind of parallel development. Now, when the role of the library is to be much closer to

the reality and the MML itself is just one among many mathematical repositories, the situation is significantly different.

Table 4. Number of accepted submissions to the MML by year

Year	1989	90	91	92	93	94	95	96	97	98	99
Articles	65	136	46	48	33	33	35	57	39	47	65

Year	2000	01	02	03	04	05	06	07	08	09
Articles	54	33	42	54	80	48	25	40	47	17

As it can be seen, the first two years were extremely fruitful. No doubt, the first one was most influential, when the fundamentals, such as basic properties of sets, relations, and functions, the arithmetic, and vector spaces were established – to enumerate among many these most important. Some articles from that time were more or less straight translations from those written in older dialects of the Mizar language (Mizar PC, Mizar-2 etc., see [9]). Especially the subsequent year – 1990, when many authors could benefit from introducing the basics, hence they were able to work on various topics in parallel, brought into the Mizar Mathematical Library the biggest number of submissions so far (136 per year), then the number stabilized.

4.4 Towards Concrete and Abstract Mathematics

As it was announced in 2001 [12], the MML will be gradually divided into two parts. As the library is based on the Tarski-Grothendieck set theory, the part devoted to the set theory (and related objects, as relations, functions, etc.) is indispensable. There is, however, huge amount of knowledge for which set theory is essential, but actually based on the notion of structures by means of the Mizar language.

There are three parts of the Mizar Mathematical Library:

- concrete, which does not use the notion of structures (here of course comes standard set theory, relations, functions, arithmetic and so on);
- abstract, i.e. `STRUCT_0` and its descendants, operating on the level of Mizar structures; both parts are not completely independent – here the concrete part is also reused (abstract algebra, general topology including the proof of the Jordan Curve Theorem, etc.);
- SCM, the part in which the theory of Random Access Turing Machines are modelled, i.e. a mathematical model of a computer is described – from here in fact all lemmas of a more general character are taken to the other two parts. This will obviously never be self-contained but probably the best separated part of the MML.

This division is reflected in the file `mml.lar` in the distribution containing the order of processing of articles (it is especially important when creating the Mizar

data base) – the "concrete" articles go first, at the end we have those devoted to the SCM series. The process of separating these three parts is very stimulating for the quality of the Mizar library – many lemmas are better clustered as a result of this activity. Paradoxically, the more articles the concrete part consists of, the better is the organization of the library, because although structures allow for more feasible formal apparatus, still many useful "concrete" lemmas are contained in the abstract part.

As of the middle of 2009, this division can be summarized in Table 5.

Table 5. The three parts of the MML

Part	Number of articles	% of total
concrete	275	25.87
abstract	733	68.96
SCM	55	5.17
Total	1063	100.00

Note that apart from the revisions suggested by the referees when giving D-grades, any user can suggest changes via the Mizar TWiki site; he may of course also send an improved version via email to the Library Committee; as an example, lemmas needed for the Gödel Completeness Theorem were reformulated to provide its better understanding as a result of the external call.

4.5 Reviewing Software

Most of massive changes of the Mizar repository are done automatically with the help of computer programs. They can be roughly divided into four groups:

- existing, publicly available in every distribution of the system (the author is encouraged to clean his article until all these tools will report no errors, but there are only a few programs of this kind):
  - on the syntactical level – `chklab`, `inacc` (removing unused labels and blocks of text), here also (`renthlab`) can be listed – unifying labels in the article; this of course does not report any errors, but it makes the source more readable as the enumeration of theorems within the Mizar file reflects that of the abstract file (without proofs),
  - on the proof level – `relprem`, `relinfer`, `reliters`, `trivdemo` (suggesting unnecessary premises, proof steps, parts of iterative equality, and nested proofs, respectively),
- with use limited to the Library Committee (here the number of tools is much bigger, including editing versions of the above) – mainly syntactical (`corrolla`, `incassum`, `irrcase`, `irrvar`, `irrpred` – pointing out theorems justified only by library references, unused assumptions, cases in proofs, irrelevant variables and local predicates, to name only a few), also cleaning and sorting the directives in the environment declaration of the article,

– based on Bancerek's MML Query [3],
– developed by Josef Urban, we will list some of them in the next section.

4.6 MML Management

As a first tool of collaborative work on the library we can enumerate Mizar TWiki (`wiki.mizar.org`) which gradually changes its profile from an experimental – and rarely used – forum into the place where suggestions/experiences with the MML can be described. In our opinion, current Wiki for Mizar is far from being attractive (e.g. it lacks the option of remote MML database building to check if the revision can affect bigger part of the library) and the application of the others' work (also based on Wiki, like that of Coq [4]) would be interesting and highly desirable. But without any additional optimizations (there is no `make` for the MML) it would be abuse to call remote processing "interactive" because as of now, the verification of the whole library takes about six hours while the regeneration of the database files is ca. 30 minutes (frankly speaking, too much of this time is just file handling).

A the most important, and probably one of the better known MML tools, we can point out MML Query [3]. It has proved its feasibility in situations when one tries to search not only on the text level (by `grep`-like tools), but to find out items which use restricted set of constructors or notations, and it is how subsequent EMM (Encyclopedia of Mathematics in Mizar) items were created, e.g. `XCMPLX`, `XREAL`, `XXREAL`, and `XBOOLE` series, dealing with complex, real, extended real numbers, and boolean properties of sets, respectively. Also researchers, when writing their Mizar articles, can find it very useful. But usually, a typical author does not care too much if his lemma which takes some ten lines of Mizar code is already present in the library. Actually, searching for such auxiliary facts can take much more time than just proving them. This results in many repetitions in the library MML Query cannot cope with. And although the author can feel uncomfortable with multiple hits of the same fact, annotating such situations and reporting it to the MML developers is usually out of his focus.

This is the area where another tool comes in handy. Potentially very useful for the enhancement of the MML as a whole, MoMM (Most of Mizar Matches) developed by Josef Urban was primarily developed to serve as an assistant during authoring Mizar articles [15]. It is a fast tool for fetching matching theorems, hence existing duplications can be detected and deleted from the MML (according to [15], more than two percent of main Mizar theorems is subsumed by the others). The work with the elimination of these lemmas is still to be done – many of detected repetitions are useful special cases (or important proof steps, as in the Jordan Curve Theorem polygonal case was), so their automatic removal is at least questionable. The authors often want to add straightforward consequences of some theorems from the MML to enrich the theory they develop and to have a complete set of properties in a single place.

Another popular software, MML CVS – the usual concurrent version system for the MML was active for quite some time, but then was abandoned because the changes were too cryptic for the reader due to the lack of proper marking of items. The main obstacle in the current state of the MML is that still there

are no absolute names for definitions and theorems. Albeit the enumeration of theorems within one file is not a big deal, especially if some of the theorems which are before the chosen are moved elsewhere (`canceled` keyword which serves as a placeholder for deleted theorem keeps the numbers right). The real problem is with the movements of bigger parts between the articles, where simple translation of the old library reference into the new one is insufficient; the user is obliged to add some environment declaration items. Then the changes are usually too massive and too technical to find out what really matters (and then it is better to check the differences on the level of the abstract – i.e. without proofs).

5 Which Way to Go?

Contemporary standards of the publishing process open some new possibilities – there are many journals online, Springer also announces their books/proceedings at their webpage. Paper-printed editions have some obvious limitations, vanishing for electronically stored and managed repositories of knowledge. We can notice, as an example, new functionalities of [10] in comparison to (even online) version of Abramowitz and Stegun [1].

5.1 Flexibility

Of course the content, once published on paper, is fixed. We can mind some real-life situations – rough sets as an example of obtaining new results via a kind of revision process (originally considered to be classes of abstraction with respect to some equivalence relation, then some of the properties were dropped to generalize the notion – see [7], [8]); also Robbins algebras and related axiomatizations of algebras are a good example, when a classical problem could be rewritten and reused when solved. In the aforementioned examples these were reasons for writing other papers, within the computerized repository the enhancement (the generalization of results) can be obtained via the revision process. Still, the problem of authorship of such "mixed", in a sense, results remains.

5.2 Distribution

As a rule, building an extensive encyclopedia of knowledge needs some investment; on the one hand, it can be considered by purely financial means as "information wants to be free, people want to be paid" [2]. That is the way Wolfram MathWorld [17] has been raised, as a collection closed in style, in fact authored by one person, Eric Weisstein.

But right after this service has been closed due to the court injunction, it soon appeared that the need to bridge this gap is that strong – many volunteers were working to develop a concurrent service to that of Wolfram's, but of the more open type, based on the mechanism similar to Wiki.

The effort of PlanetMath[5] is now a kind of Wikipedia for mathematics (in fact they even cooperate closely). Adding or editing content by virtually anyone is an

[5] `http://planetmath.org`.

obvious advantage of such sources, but at the same time this also makes them less stable and less reliable. Jimmy Wales, Wikipedia's founder, recalls that it is not always a definitive source of information, and hence is not ideal for academic purposes, if the knowledge is not supported by books, journals, etc. The urgent need for correctness checking becomes critical when mathematical knowledge is taken into account. Here, however proof assistants give the valuable possibility of checking, at least for the correctness of proofs; still the question of whether the definitions are right, i.e. how the encoded version reflects real mathematical objects, is the one only humans can answer to a full extent.

As we can observe based on the HOL Light system, John Harrison is the person who formalized all 74 theorems from Wiedijk's "Top 100 Mathematical Theorems" [16] proven with this proof-assistant. In the case of Mizar, 50 were developed by 36 people (notable exception is Marco Riccardi who during past three years formalized twelve important facts from the list). Archive of Formal Proofs of Isabelle is closer to the MML in this sense. Of course, for the bigger number of developers, high price of lacking homogeneity must be paid. It is important then, to have a group of supervisors (the "core team") of the repository, as is the Library Committee for the MML.

5.3 The Question of Authorship

Even in non-profit projects, like GNU, people may want to get their payment in another form: at least the added annotation such as "This article is owned by...", as in PlanetMath, which can also be considered a kind of motivation to keep higher standards of the encyclopedia since the authorship is not fully anonymous.

In the MML the authorship is fixed (every article is annotated, there are some items of Library Committee), there were however, especially recently, cases when the parts of submissions moved around so that the authorship actually exchanged (as for example, with the formalization of the Zorn Lemma, originally created by Grzegorz Bancerek, and now, after the changes concerned with the move of this article to the concrete part, attributed to Wojciech Trybulec). Of course, the content published in the automatically translated journal *Formalized Mathematics* is authored accordingly with the original, not revised version. In the case of many articles, especially those submitted much earlier, the content which was written by their author is only a small part of the original size due to new language capabilities or the generalizations which caused triviality of theorems proven there. There is over 200 authors of MML items; many of them are no longer active (they left the Mizar project), some were just short-term collaborators (e.g., students who did the work suggested by someone else) and the ownership of such article does not matter so much, at least for the authors.

It is clear that databases of knowledge will not be (so is not the MML as of now) only collections of formalized existing, classical results. The authors who develop their own results should retain their rights. Partly, although not officially stated, it is achieved via dividing the article into two parts: preliminary, or of much more general interest, and the proper submission. The first section is meant to be moved eventually in a more appropriate place in the core of the MML. Usually

the reviewers' opinions reflect this policy – and even if the grade D is not given explicitly, the suggestions are expected to be taken into account by the Library Committee.

6 Final Remarks

To meet the expectations of researchers being potential users of repositories of mathematical knowledge, such collections cannot be frozen. The availability of the contemporary electronic media opens new directions of the development of the new encyclopedias yet unavailable for their paper counterparts. The need of the enhancement stems not only from the fact that there may be some obvious mistakes in the source; the reasons are far more complex.

In the paper, we tried to point out some of the issues connected with the mechanism of revisions performed on the Mizar Mathematical Library, a large repository of computer-verified mathematical knowledge. The dependencies between its items and the environment declaration (notation and especially, constructors) are as of now too complex to freely move a single definition or a theorem between separate articles.

In our opinion, the current itemization of the MML into articles does not fit the needs we expect from the feasible repository of mathematical facts; if we try to keep authors' rights unchanged, there is an emerging need to have some other, smaller items which guarantee the developer's authorship rights, a kind of ownership similar to that used in the PlanetMath project. We propose to keep the authorship for any basic Mizar block/item (theorems, definitions and registrations), having in mind that some of private lemmas which are not exported to the data base, can be significant steps of the proof of a public theorem.

Also the better automation of the MML revision process is strongly desirable. However possible, at least to some extent, due to some difficulties which can be met as we pointed out, the human supervision of such automatic changes will probably always be needed.

Acknowledgments

The first author wants to express his gratitude to Andrzej Trybulec and Artur Korniłowicz for their continuous cooperation on the enhancement of the MML. We express also our gratitude to all the reviewers of the Mizar Mathematical Library (especially Adam Naumowicz) for their great job done.

References

1. Abramowitz, M. and Stegun, I.A.: *Handbook of Mathematical Functions*; National Bureau of Standards, Applied Mathematics Series No. 55, U.S. Government Printing Office, Washington, DC, 1964, see also `http://www.convertit.com/Go/ConvertIt/Reference/AMS55.ASP`.

2. Adams, A.A. and Davenport, J.H.: *Copyright issues for MKM*, in: A. Asperti, G. Bancerek, and A. Trybulec (eds.), Proc. of MKM 2004, LNCS 3119, Springer, pp. 1–16, 2004.
3. Bancerek, G.: *Information retrieval and rendering with MML Query*; in: J.M. Borwein and W.M. Farmer (eds.), Proc. of MKM 2006, Lecture Notes in Artificial Intelligence 4108, pp. 65–80, 2006.
4. Corbineau, P. and Kaliszyk, C.: *Cooperative repositories for formal proofs*; in: M. Kauers, M. Kerber et al. (eds.), *Towards Mechanized Mathematical Assistants,* Proc. of MKM 2007, LNAI 4573, Springer, pp. 221–234, 2007.
5. de Bruijn, N.G.: *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.
6. Davenport, J.H.: *MKM from book to computer: A case study*; in: A. Asperti, B. Buchberger, and J. Davenport (eds.), Proc. of MKM 2003, LNCS 2594, Springer, pp. 17–29, 2003.
7. Grabowski, A.: *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.
8. Grabowski, A. and Schwarzweller, C.: *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, pp. 130–144, 2004.
9. Matuszewski, R. and Rudnicki, P.: MIZAR*: the first 30 years*; Mechanized Mathematics and Its Applications, 4(**1**), pp. 3–24, 2005.
10. Miller, B.R. and Youssef, A.: *Technical aspects of the Digital Library of Mathematical Functions*; Annals of Mathematics and Artificial Intelligence, 38, pp. 121–136, 2003.
11. The Mizar Homepage; `http://www.mizar.org/`.
12. Rudnicki, P. and Trybulec, A.: *Mathematical Knowledge Management in Mizar*; in: B. Buchberger and O. Caprotti (eds.), Proc. of MKM 2001, Linz, Austria, 2001.
13. Rudnicki, P. and Trybulec, A.: *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, pp. 162–174, 2003.
14. Sacerdoti Coen, C.: *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, pp. 30–44, 2003.
15. Urban, J.: *MoMM – fast interreduction and retrieval in large libraries of formalized mathematics,* International Journal on Artificial Intelligence Tools, 15(1), pp. 109–130, 2006.
16. Wiedijk, F.: *Formalizing 100 Theorems,* `http://www.cs.ru.nl/~freek/100/`.
17. Wolfram Mathworld web page; `http://mathworld.wolfram.com/`.

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 18 (31) 2009

A Language for Mathematical Knowledge Management

Steven Kieffer[1], Jeremy Avigad[2], and Harvey Friedman[3⋆]

[1] Simon Fraser University
[2] Carnegie Mellon University
[3] Ohio State University

Abstract. We argue that the language of Zermelo Fraenkel set theory with definitions and partial functions provides the most promising bedrock semantics for communicating and sharing mathematical knowledge. We then describe a syntactic sugaring of that language that provides a way of writing remarkably readable assertions without straying far from the set-theoretic semantics. We illustrate with some examples of formalized textbook definitions from elementary set theory and point-set topology. We also present statistics concerning the complexity of these definitions, under various complexity measures.

1 Introduction

With the growing use of digital means of storing, communicating, accessing, and manipulating mathematical knowledge, it becomes important to develop appropriate formal languages for the representation of such knowledge. But the scope of "mathematical knowledge" is broad, and the meaning of the word "appropriate" will vary according to the application. At the extremes, there are competing desiderata:

- At the *foundational level*, one wants a small and simple syntax, and a precise specification of its semantics. In particular, one wants a specification as to which inferences are valid.
- At the *human level*, one wants to have mathematical languages that are as easy to read and understand as ordinary mathematical texts, yet also admit a precise interpretation to the foundational level.

For ordinary working mathematicians, the foundational interpretation is largely irrelevant, but some sort of formal semantics is necessary if the information encoded in mathematical texts is to be used and manipulated at the formal level. Of course, one solution is simply to pair each informal mathematical assertion with a formal translation, but then there is the problem of obtaining the formal translations and ensuring that they match the intention of the informal text. As a result, it is

⋆ Work by Avigad and Friedman partially supported by NSF grant DMS-0700174.

ISBN 978-83-7431-229-5 ISSN 0860–150X

more promising to use semi-structured languages that integrate features of both the foundational and human levels. This results in a smooth spectrum of languages in between the two extremes. At intermediate "expert user" levels, one may want a language whose structure is close to that of the underlying foundational framework, yet is as humanly readable as possible.

To complicate matters, there are features of mathematical knowledge that are not captured at the level of assertions: mathematical language is used to communicate definitions, theorems, proofs, algorithms, and problems, among other things. At the level of a mathematical theory, language is also used to communicate relationships between these different types of data. The formal information that is relevant will vary depending on the application one has in mind, be it database access and search, theorem proving, formal verification, etc.

Here we will be primarily concerned with mathematical assertions as they are used to state definitions and theorems.[4] If one is looking for a foundational framework that is robust enough to subsume those used by most systems of MKM, it is hard to beat the language of set theory: we know of no foundational system other than Quine's New Foundations that cannot be interpreted in the language of set theory in such a way that inferences are reduced to inferences in Zermelo-Fraenkel set theory with the axiom of choice (ZFC), or some plausible extension (say, with large universes of sets). To be clear, we are not denying the importance of other frameworks for more specific purposes. For example, the theory of real closed fields is appropriate to representing many constraint problems, and constructive frameworks are better suited to certain forms of algorithmic reasoning. It is also important to find ways of sharing the additional information that comes with the use of these more restricted frameworks. We are simply singling out set theory as a unifying framework for expressing what assertions in the various local frameworks have in common.

We extend the foundational framework in two ways. First, we allow for explicit definitions of new predicates and functions on the universe of sets. And, second, we allow function symbols to denote functions that are only partially defined, using a logic of partial terms. We call the resulting formal system $DZFC$. As we observe in Section 2, this system is easily shown to be conservative over ZFC. We argue that these extensions are *not* just a matter of syntactic sugar, but, rather, are essential to adequate representation of the mathematical data: there is a difference between assertions using defined terms and their expanded versions, and, in mathematical terms, $1/0$ really is an undefined quantity. Thus $DZFC$ is our proposal for a foundational language and its semantics.

Our main goal here is to show that the distance between this foundational level and ordinary mathematical text is not as far as is commonly supposed, by presenting a syntactically-sugared version of set theory, PST, that is simultaneously

[4] In passing, we note that computational proof assistants like Mizar [13], HOL [7], Isabelle [12], Coq [3] and HOL light [8] all provide languages that can be used to describe mathematical proofs. Of these, the Mizar and Isabelle/Isar languages model human proof languages most closely. The Isar effort [16] shows that the proof language is somewhat orthogonal to the assertion language; that is, Isar can be instantiated to various foundational frameworks, subject only to minor constraints.

close to both. On the one hand, we show that our language is easily parsed and translated to *DZFC*. On the other hand, by automatically replacing symbolic expressions with user-provided natural language equivalents, we obtain output that is humanly readable, and, although not exactly literary, recognizably faithful to the original mathematical texts.

We support this last claim with examples from Suppes's *Axiomatic set theory* [14] and Munkres's *Topology* [11]. In each case, we present our formal input with both *DZFC* and our natural language translations. Indeed, the appendices to Kieffer [9] provide a corpus of 341 definitions, taken from Chapters 2–6 of Suppes and Sections 12–38 of Munkres. Examples of the natural language translations can be found in Appendix B, below. These examples show that *PST* offers a promising target semantics for mathematical markup languages, like OMDoc [10].

To illustrate the utility of *PST*, we describe two pieces of software that take advantage of both the formal structure of the definitions and their proximity to the informal text. First, we describe statistical studies of the complexity of definitions in our corpus, measured in various ways. Our analysis shows, not surprisingly, that expanding definitions to the pure language of set theory yields formulas that are huge. Perhaps more surprisingly, quantifier complexity of definitions remains remarkably low, even when they are expanded to *DZFC*. We also describe software that makes it possible to explore definitional dependencies, expanding and compressing nodes via a graphical interface. To be sure, data like this can be mined from contemporary formal verification efforts.[5] But mathematical developments are often changed significantly in the process of formalization; what distinguishes the data presented here is the extent to which it faithfully represents the informal texts it is supposed to model.

Our "user-friendly" version of set theory is based on Friedman [6]; see also an earlier version in Friedman [5]. Most of the work described here, including the implementation of the parser, the entering of the data from Suppes's and Munkres's books, and associated software, constitute Kieffer's MS thesis [9], written under Avigad's supervision. The thesis and code described here, as well as additional samples of the natural language translations, can be found via Avigad's web page.[6]

2 *ZFC* with definitions and partial terms

It is widely acknowledged that Zermelo-Fraenkel axiomatic set theory with the axiom of choice, *ZFC*, is robust enough to accommodate ordinary mathematical arguments in a straightforward way. The most notable exceptions are category-theoretic arguments which rely on the existence of large universes with suitable closure properties; but these can be formalized in extensions of *ZFC* with suitable large cardinal axioms, or by restricting the closure properties of the universes in question.

[5] See, for example, the MPTP challenges, `http://www.cs.miami.edu/~tptp/MPTPChallenge/`.

[6] Specifically, see `http://www.andrew.cmu.edu/user/avigad/Papers/mkm/`.

In this section, we describe a conservative extension *DZFC* of *ZFC*. This theory incorporates two features that allow for a more direct and natural mathematical modeling:

- it accommodates partially defined functions, and hence undefined terms; and
- it allows the introduction of new function and predicate symbols to stand for explicitly defined functions and predicates.

We describe each of these extensions, in turn.

To start with, *DZFC* is based on a free logic, with a special predicate $E(t)$. This is usually written $t{\downarrow}$, and can be read "t is defined" or "t denotes." The axioms governing the terms are presented as the "logic of partial terms" in Beeson [2], E^+ logic in Troesltra and Schwichtenberg [15]; see also the very helpful explanation and overview in Feferman [4]. The basic idea is that variables in the language range over objects in the intended domain (in our case, sets), but, as function symbols may denote partial functions, some terms fail to denote. So, for example, the axioms for universal instantiation are given by $\forall x\ \varphi(x) \wedge t{\downarrow} \rightarrow \varphi(t)$. The basic relation symbols of *ZFC*, which we take to be $\in$ and $=$, are assumed only to hold between terms that denote; thus we have axioms $s \in t \rightarrow s{\downarrow} \wedge t{\downarrow}$ and $s = t \rightarrow s{\downarrow} \wedge t{\downarrow}$. Partial equality $s \simeq t$ is defined as usual by the axiom $s \simeq t \leftrightarrow (s{\downarrow} \vee t{\downarrow} \rightarrow s = t)$.

Next, the syntax of ordinary set theory is extended to include definition descriptions, *à la* Russell. Formally, for each formula $\varphi(x)$, the expression $(\iota x)\varphi(x)$ is a term whose free variables are just those of φ, other than x. These terms are governed by the axioms

$$y = (\iota x)\varphi(x) \leftrightarrow \forall z\ (\varphi(z) \leftrightarrow z = y).$$

Thus in *DZFC* one can show that $(\iota x)\varphi(x)$ is defined if and only if there is a unique y satisfying $\varphi(y)$, in which case, $(\iota x)\varphi(x)$ is equal to that y.

Finally, one is allowed to introduce new function symbols and relation symbols to abbreviate formulas and terms. That is, for each formula $\varphi(x, \bar{y})$, one can introduce a new function symbol $f(\bar{y})$ with the axiom

$$f(\bar{y}) \simeq (\iota x)\varphi(x, \bar{y}),$$

and for every formula $\psi(\bar{y})$ one can introduce a new relation symbol $R(\bar{y})$ with the axiom

$$R(\bar{y}) \leftrightarrow \psi(\bar{y}).$$

It is not hard to show that adding the usual axioms of set theory to this framework yields a conservative extension:

Theorem 1. *DZFC is a conservative extension of ZFC.*

The proof amounts to an interpretation of partial functions and elimination of definitions that is by now standard; details can be found in [15, 9]. Note, however, that the usual method of eliminating defined function symbols and relation symbols by replacing them by their definiens can result in an exponential increase in length.

3 The language of practical set theory, *PST*

We now describe a more flexible language, *Practical set theory*, or *PST*, designed by Friedman. This language has two key features:

- The language incorporates a healthy amount of syntactic sugar, making it possible to express ordinary mathematical definitions and assertions in a natural way.
- The language is easily and efficiently translatable to *DZFC*.

In this section we describe some of the features of *PST* and the translation to *DZFC*. A full and precise specification of the *PST* and its *DZFC* semantics can be found in [9, 6], where it was called the *Language of Proofless Text*, or *LPT*. The claims of naturality will be supported with examples in the next section and in Appendix B.

The starting point for *PST* is the usual syntax of first-order logic. We adopt conventions to distinguish between variables, defined functions, and relations; application of a defined relation *REL* to terms $t_1, \ldots, t_k$ is written with square brackets $REL[t_1, \ldots, t_k]$, while application of a defined function *Fun* is written with parentheses, $Fun(t_1, \ldots, t_k)$. The usual language of first-order logic is augmented with a significant amount of "syntactic sugar," to make the expression of mathematical notions as convenient as possible. These include the following.

Function application for sets. Any term may be used as though it were a function, of any arity (including "infix"). For example, one may quantify a variable f, and then proceed to use it as though it were a function. In *PST*, $f(x)$ denotes the unique u such that the ordered pair $\langle x, u\rangle$ is in f, assuming there is such u. The following definition of the unary predicate `FCN` therefore asserts that f is a function if it is a set of ordered pairs $\langle x, u\rangle$ in which no x occurs more than once as the first component of a pair.

DEFINITION FS.2.58: 1-ary relation `FCN`. $\texttt{FCN}[f] \leftrightarrow f = \{\langle x, y\rangle : f(x) = y\}$.

Finite sets and tuples. In the previous example, we saw a finite tuple; namely, the ordered pair $\langle x, y\rangle$. Tuples of any finite length are terms in *PST*.

A finite set can be denoted by simply listing all of its elements. For example, in defining the Wiener-Kuratowski ordered pair, we may use the term $\{\{a\}, \{a, b\}\}$.

Set-builder notation. The example above illustrates the use of set-builder notation. In *PST*, the term $\{t : \varphi\}$ denotes the set of all values of $t(x_1, \ldots, x_n)$, where the variables $x_1, \ldots, x_n$ occurring in t range over tuples satisfying $\varphi(x_1, \ldots, x_n)$. Note that this involves an essential use of partiality; for example, in the intended semantics, the term $\{x : x = x\}$ is undefined.

Suppose we wish to define $\texttt{Image}(f)$ to be the set of all $f(x)$ such that $x \in \texttt{Dom}(f)$. The expression

$$\texttt{Image}(f) \simeq \{f(x) : x \in \texttt{Dom}(f)\}$$

is not what we want, because f on the right-hand side is taken to be a bound variable ranging over the universe of sets. Instead, *PST* has us write

$$\texttt{Image}(f) \simeq \{f(x) : x \in \texttt{Dom}(f),\ f \text{ fixed}\}$$

to indicate that the expression depends on a fixed value of f.

Defined function symbols. We use an exclamation mark in place of Russell's ι as a definite description operator. It is used in the next example, where we define an infix function, $+_{\mathbb{Q}}$, for addition on the rational numbers. Every infix function is given a *precedence* number, for use in determining order of operations.

DEFINITION FS.5.25: Infix function $+_{\mathbb{Q}}$. $x +_{\mathbb{Q}} y \simeq (!z)(x, y, z \in \mathbb{Q} \wedge (\exists a, b, c)$ $(a \in x \wedge b \in y \wedge c \in z \wedge a +_{SUB} b = c))$. Precedence 40.

A definition may be composed of any number of "If ... then ..." clauses, and may end with one "Otherwise ..." clause, which allows definition by cases, as in the example below. In this example the 'Otherwise' clause introduces a condition under which the function is undefined. For this we use the predicate $\uparrow$, and this allows for the definition of partial functions.

DEFINITION FS.2.3: 1-ary function $\texttt{Dom}$. If $\texttt{BR}[R]$ then $\texttt{Dom}(R) \simeq \{x : (\exists y)(x\ R\ y)\}$. Otherwise $\texttt{Dom}(R)\uparrow$.

Defined relation symbols. As with functions, we may define infix relations, as in the definition of $<$ on the rational numbers, below.

DEFINITION FS.5.24: Infix relation $<_{\mathbb{Q}}$. $x <_{\mathbb{Q}} y \leftrightarrow (\exists z, w)(x, y \in \mathbb{Q} \wedge z \in x \wedge w \in y \wedge$ $z <_{\texttt{SUB}} w)$.

Lambda notation. PST includes a lambda operator which can be used to bind variables and thereby denote functions. In the example below, we define a binary function called `Cartespow` (for "Cartesian power"). This function maps a pair of sets A, B to the set A^B; i.e., a product of B-many copies of A. The definition relies on a previously defined function, `Cartesprod` (for "Cartesian product"), a binary function taking a map f and a set C to the product over $c \in C$ of the sets $f(c)$. The definition of `Cartespow` uses lambda abstraction to define the constant function $b \mapsto A$ on the fly, to serve as the first argument to `Cartesprod`.

DEFINITION MunkTop.19.2.5: 2-ary function $\texttt{Cartespow}$. $\texttt{Cartespow}(A, B) \simeq$ $\texttt{Cartesprod}((\lambda b \in B)(A), B)$.

Infix relation chains. Infix relations may be chained together in the usual way, as with the $<_{\mathbb{R}}$ relation in the example below.

DEFINITION MunkTop.13.3.a.basis: 0-ary function `Stdrealtopbasis`.
`Stdrealtopbasis` $\simeq \{U \subseteq \mathbb{R} : (\exists a, b \in \mathbb{R})(U = \{x \in \mathbb{R} : a <_{\mathbb{R}} x <_{\mathbb{R}} b\})\}$.

Bounded quantifiers. Quantified variables and variables used in set-builder notation may be bounded by any infix relation, as in the example above.

The translation from *PST* to *DZFC* is not difficult. Since our grammar for *PST* is not LL, we used the ACCENT compiler-compiler [7], which implements Earley's algorithm. The latter can parse any context-free grammar in cubic time, and runs in quadratic time when the grammar is unambiguous [1].

Appendix A contains a number of examples of *PST* definitions, together with their translations to *DZFC*. In each case, we present the *DZFC* input, a LaTeX representation of that input generated by the parser, and the translation to *DZFC*. A much larger corpus of examples – 183 definitions from Suppes's *Axiomatic Set Theory* [14] and 148 definitions from Munkres's *Topology* [11] – can be found in [9]. In practice, the translation took at most a few seconds to process a file containing a dozen large definitions. Comparing the (LaTeX) *DZFC* output with the (LaTeX version of the) *PST* input yields a factor of about 0.91, which is to say, the *DZFC* translations are actually slightly shorter.

4 Natural language output

The examples of *PST* input in the last section are readable, but not attractive. It is hard to remember meaning of symbols "BR" or "TOPSP"; it would help to have phrases like "is a binary relation" or "is a topological space." In fact, even for logical connectives like $\wedge$, natural language equivalents like "and" are generally easier to read. In an ordinary mathematical language text, however, words are not always favored over symbols. For example, defined functions are usually given symbols: $gcd(x, y)$ instead of "the greatest common divisor of x and y." Binary relations like $=$ and $<$ are usually preferred to "equal to" and "less than." On the other hand, unary relations often represent concepts that are expanded to words, as shown by the examples above.

In light of these observations, we chose to output natural language equivalents for the connectives, and allow the user to input natural language equivalents for defined symbols. For example, with the entry

```
TOPSP:2@
  reln:$(#^0,#^1)$ is a %e?topological space%ee?@
  negn:$(#^0,#^1)$ is not a topological space@
  plur:%$(#^0,#^1)$% are topological spaces@
  nplu:%$(#^0,#^1)$% are not topological spaces@@
```

[7] `http://accent.compilertools.net/`.

the user can specify the natural language that should be used in place of the `TOPSP` relation.

In some cases, either symbols or a natural language equivalent can be used, as in $\{x \in \mathbb{N} \mid \ldots\}$ or "the set of $x \in \mathbb{N}$ such that ...." It is usually awkward to have natural language occur as a subterm of a symbolic expression; for example, consider "1+the greatest common divisor of x and y." Thus we incorporate a monotonicity rule: once a subterm of a term has been expanded to natural language, natural language versions are favored from then on. This choice yields, for example, $\{x \in \mathbb{N} \mid a < x < b\}$, but also "the set of x in $\mathbb{N}$ such that $a < x < b$ and x is even."

Accordingly, the user supplies two clauses for a defined function or relation for which symbols are preferred over words:

```
\wp:1@
  symb:$\wp(#^0)$@
  word:the power set of #0@@
```

whereas if words are the desired default then just one clause is needed:

```
Stdrealtop:0@
  word:the standard topology on $\mathbb{R}$@@
```

Appendix B provides examples of natural language output. We emphasize that these were generated directly from the *PST* input, using the additional natural language data, supplied by the user, described above. Although the definitions are not exactly literary, they are surprisingly readable, and close to ordinary mathematical text. It is certainly the case that additional heuristics could be used to render the output more attractive, and additional markup from the user would result in improvements. In other words, there is a lot more that can be done along these lines; our claim here is only that *PST* offers an auspicious start.

5 Exploring definitions

Among the benefits of having a database of definitions is the ability to explore those definitions interactively. We designed two simple programs with which to demonstrate some of the possibilities.

Our first program allows the interactive display and manipulation of directed acyclic graphs (dags) of conceptual dependencies, as depicted in Figure 1.

With a second program we gathered statistics on these graphs. Associated to each definition is the dag of all definitions on which it depends; by the *size* of this dag we mean the number of vertices, and by the *depth* of this dag we mean the length of its longest directed path. Table 1 shows the maximum and mean values for all definitions in our database.

Additional statistics, including data on the quantifier complexity of definitions in our corpus, can be found in Appendix C.

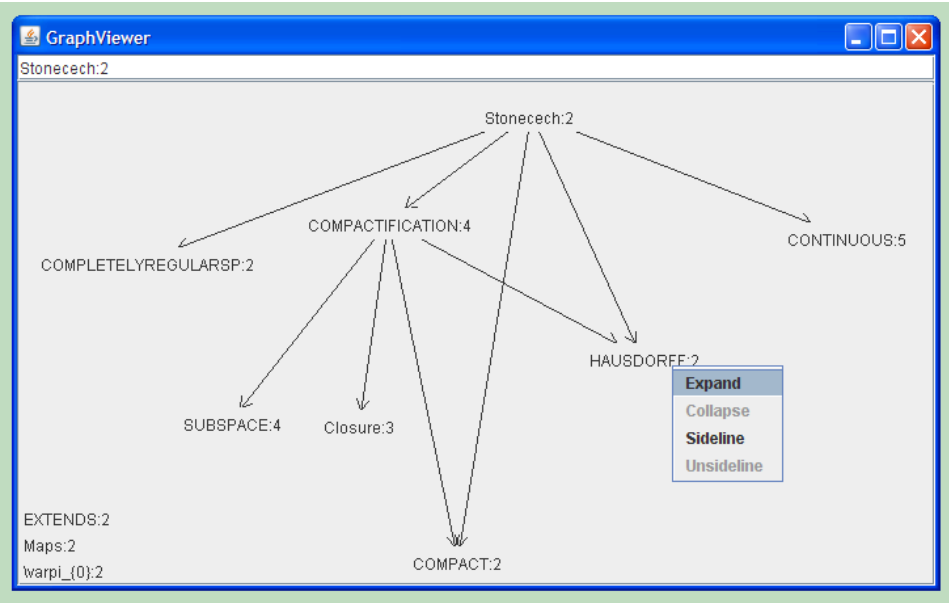


Fig. 1. Exploring the definition dag for the Stone-Čech compactification.

Table 1. Max and mean dag sizes and depths

		Max	Mean
All	Depth	32	10.77
	Size	110	29.56
Suppes	Depth	26	10.09
	Size	77	25.91
Munkres	Depth	32	12.25
	Size	110	36.01

6 Conclusions

We have argued that one should adopt a language close to definitional set theory as a uniform language to support communication and exchange of mathematical results. The particular language we describe here, *Practical set theory*, fares well in that regard: it is easy and natural to work with, providing a high-degree of readability while remaining close to a clear foundational semantics.

Appendix A: Examples of *PST* input and *DZFC* translations

We consider a few examples of formal definitions, highlighting the naturality of *PST* over *DZFC*. (The ϖ_0 function appearing in the *DZFC* translations is a function defined to take (a, b) to the Wiener-Kuratowski ordered pair $\{\{a\}, \{a, b\}\}$.)

Example 1. Here the description operator is used in *PST* to bind an ordered pair, so that we are able to refer to "the unique ordered pair $\langle Y, T' \rangle$ such that...." This translates to a much clumsier expression in *DZFC*, requiring two additional bound variables.

PST input:

```
DEFINITION MunkTop.29.4: 2-ary function Oneptcompactification.
If TOPSP[X,T] then Oneptcompactification(X,T) \simeq
(!<Y,T'>)(
  COMPACTIFICATION[Y,T',X,T] \wedge Y \less X \approx_{C} 1_{N}
).
```

PST rendered in LaTeX:

DEFINITION MunkTop.29.4: 2-ary function `Oneptcompactification`. If `TOPSP`$[X, T]$ then `Oneptcompactification`$(X, T) \simeq (!\langle Y, T' \rangle)$ (`COMPACTIFICATION`$[Y, T', X, T] \wedge Y \backslash X \approx_{\mathtt{C}} 1_{\mathtt{N}}$).

DZFC translation:

`Oneptcompactification`$(X, T) \simeq (\iota y_0)($`TOPSP`$[X, T] \wedge y_0 \simeq (\iota x_0)(\exists Y, T')(x_0 = \varpi_0(Y, T') \wedge ($`COMPACTIFICATION`$[Y, T', X, T] \wedge \approx_{\mathtt{C}}[\backslash(Y, X), 1_{\mathtt{N}}])))$

Example 2. Next observe what happens in *DZFC*, where we cannot match the brevity of expression used in our definition of the `FCN[f]` predicate in *PST* (which says that `f` is a function).

PST input:

```
DEFINITION FS.2.58: 1-ary relation FCN. FCN[f] \iff
f = {<x,y> : f(x) = y}.
```

PST rendered in LaTeX:

DEFINITION FS.2.58: 1-ary relation FCN. $\texttt{FCN}[f] \leftrightarrow f = \{\langle x, y\rangle : f(x) = y\}$.

DZFC translation:

$\texttt{FCN}[f] \leftrightarrow f = (\iota z_0)(\forall y_0)(y_0 \in z_0 \leftrightarrow (\exists x, y)(y_0 = \varpi_0(x, y) \wedge ((\iota x_0)(\varpi_0(x, x_0) \in f) = y)))$

Example 3. Here we see how important the lambda operator is:

PST input:

```
DEFINITION MunkTop.19.2.5: 2-ary function Cartespow. Cartespow(A,B)
\simeq Cartesprod((\lambda b \in B)(A),B).
```

PST rendered in LaTeX:

DEFINITION MunkTop.19.2.5: 2-ary function Cartespow. $\texttt{Cartespow}(A, B) \simeq \texttt{Cartesprod}((\lambda b \in B)(A), B)$.

DZFC translation:

$\texttt{Cartespow}(A, B) \simeq \texttt{Cartesprod}((\iota z_0)(\forall y_0)(y_0 \in z_0 \leftrightarrow (\exists b, x_0)(y_0 = \varpi_0(b, x_0) \wedge x_0 = (A) \wedge b \in B)), B)$

Appendix B: Examples of the natural language translations

In some cases our natural language generating program pst2nl produces output that is quite close to what a human being might write. For example, from the following *PST* input,

DEFINITION MunkTop.13.2: 2-ary function Basisgentop. If $\texttt{TOPBASIS}[\mathscr{B}, X]$ then $\texttt{Basisgentop}(\mathscr{B}, X) \simeq (!\mathscr{T} \subseteq \wp(X))((\forall U \subseteq X)(U \in \mathscr{T} \leftrightarrow (\forall x \in U)(\exists B \in \mathscr{B})(x \in B \wedge B \subseteq U)))$.

we get the following NL (natural language) output:

Definition: If $\mathscr{B}$ is a basis for a topology on X then *the topology on X generated by $\mathscr{B}$* is the unique $\mathscr{T} \subseteq \wp(X)$ such that for every $U \subseteq X$, $U \in \mathscr{T}$ if and only if for every $x \in U$, there exists $B \in \mathscr{B}$ such that $x \in B$ and $B \subseteq U$.

Indeed, this is not substantially different from the original text in Munkres [11], page 78. After defining what it means for $\mathscr{B}$ to be a basis, Munkres says,

> If $\mathscr{B}$ satisfies these two conditions, then we define the *topology $\mathscr{T}$ generated by $\mathscr{B}$* as follows: A subset U of X is said to be open in X (that is, to be an element of $\mathscr{T}$) if for each $x \in U$, there is a basis element $B \in \mathscr{B}$ such that $x \in B$ and $B \subset U$. Note that each basis element is itself an element of $\mathscr{T}$.

What is more common is that the output of `pst2nl` reads nicely except for a "run-on" sound, resulting from insufficient punctuation. For example:

Definition: If R is a strong simple order on X then *the basis for the order topology on* (X, R) is the set of U such that there exist a, $b \in X$ such that $U = (a, b)$ or a is a first element in X and $U = [a, b)$ or b is a last element in X and $U = (a, b]$.

In Munkres, page 84, all of this information is spread out over a numbered list:

> **Definition.** Let X be a set with a simple order relation; assume X has more than one element. Let $\mathscr{B}$ be the collection of all sets of the following types:
>
> 1. All open intervals (a, b) in X.
> 2. All intervals of the form $[a_0, b)$, where a_0 is the smallest element (if any) of X.
> 3. All intervals of the form $(a, b_0]$, where b_0 is the largest element (if any) of X.
>
> The collection $\mathscr{B}$ is a basis for a topology on X, which is called the *order topology*.

Heuristics, combined with additional user markup, could eventually be incorporated to help improve the flow and punctuation of the translations. We have implemented one easy improvement already, whereby adjacent assertions of a common predicate are combined into a single assertion using plural form. Thus, from the *PST* input,

DEFINITION MunkTop.12.4.a: 3-ary relation `FINERTOP`.
If $\texttt{TOPSP}[X, \mathscr{T}] \wedge \texttt{TOPSP}[X, \mathscr{T}']$ then $\texttt{FINERTOP}[\mathscr{T}', \mathscr{T}, X] \leftrightarrow \mathscr{T}' \supseteq \mathscr{T}$.

we obtain:

Definition: If $(X, \mathscr{T})$ and $(X, \mathscr{T}')$ are topological spaces then $\mathscr{T}'$ is *finer* than $\mathscr{T}$ on X if and only if $\mathscr{T}' \supseteq \mathscr{T}$.

This time Munkres is able to make several definitions in a single paragraph, and can abbreviate a more complex logical locution with the phrase "respective situations."

> **Definition.** Suppose that $\mathscr{T}$ and $\mathscr{T}'$ are two topologies on a given set X. If $\mathscr{T}' \supset \mathscr{T}$, we say that $\mathscr{T}'$ is *finer* than $\mathscr{T}$; if $\mathscr{T}'$ *properly* contains $\mathscr{T}$, we say that $\mathscr{T}'$ is *strictly finer* than $\mathscr{T}$. We also say that $\mathscr{T}$ is *coarser* than $\mathscr{T}'$, or *strictly coarser*, in these two respective situations. We say $\mathscr{T}$ is *comparable* with $\mathscr{T}'$ if either $\mathscr{T}' \supset \mathscr{T}$ or $\mathscr{T} \supset \mathscr{T}'$.

We consider a final example,

DEFINITION MunkTop.13.3.c: 0-ary function `Krealtop`. $\texttt{Krealtop} \simeq \texttt{Basisgentop}(\texttt{Stdrealtopbasis} \cup \{V \subseteq \mathbb{R} : (\exists W \in \texttt{Stdrealtopbasis})(V = W \backslash \{\texttt{Incl}_{\texttt{FrR}}(\texttt{1}_{\texttt{N}}/n) : n \in \mathbb{N}\})\}, \mathbb{R})$.

for which the NL output is as follows:

Definition: *The K-topology on* $\mathbb{R}$ is the topology on $\mathbb{R}$ generated by the standard basis for a topology on $\mathbb{R}$ union the set of $V \subseteq \mathbb{R}$ such that there exists W in the standard basis for a topology on $\mathbb{R}$ such that $V = W \setminus \{1/n : n \in \mathbb{N}\}$.

There are two sets mentioned in this definition: the set of $V \subseteq \mathbb{R}$ such that ..., and the set of $1/n$ such that .... According to the "monotonicity rule" described in Section 4, the latter is rendered in symbols since it has no subterm in words; the former is rendered in words since its subterm, "the standard basis for a topology on $\mathbb{R}$" has no symbolic form, and is displayed in words by default.

Another feature of `pst2nl` is apparent in this last example, where the word "in" appears before "the standard basis...." We get this preposition rather than the incorrect phrase "is in," thanks to the final clause in the user-supplied natural language equivalents for the $\in$ relation:

```
\in:infix@
  symb:#0 $\in$ #1@
  nsym:#0 $\not\in$ #1@
  reln:#0 is %e?in%ee? #1@
  negn:#0 is not in #1@
  plur:%#0% are in #1@
  nplu:%#0% are not in #1@
  prep:#0 in #1@@
```

Finally we note that the user is free to suppress artifacts of formalization, in the NL output. In the *PST* above there is an inclusion function $\texttt{Incl}_{\texttt{FrR}}$, and the number 1 is subscripted as $1_{\texttt{N}}$. None of this shows up in the NL output.

Comparison with Munkres, page 82, reveals that he is free to write in a less regimented form than that of our definitions:

> Finally, let K denote the set of all numbers of the form $1/n$, for $n \in \mathbb{Z}_+$, and let $\mathscr{B}''$ be the collection of all open intervals (a, b), along with all sets of the form $(a, b) - K$. The topology generated by $\mathscr{B}''$ will be called the *K-topology* on $\mathbb{R}$.

Appendix C: Data on quantifier complexity and length

Our database of definitions entered in *PST* consists of 183 definitions from Suppes's *Axiomatic Set Theory* [14] and 148 definitions from Munkres's *Topology* [11].

Quantifier complexity data. For each definition in our database, we measured quantifier complexity in eight different ways. In the first place, we considered both alternating quantifier depth, and non-alternating. Secondly, we considered each definition in four different states: (1) as given in *PST*; (2) as translated into *DZFC*; (3) the *expanded* version of the *DZFC*, that is, with all definienda replaced by their

definiens, recursively, until the process halts; and (4) a *partially expanded* version of the *DZFC* in which certain low-level, foundational definienda were left unexpanded, namely: the union, intersection, and set difference operations, the ordered pair, and powerset functions, the empty set, and the subset and superset relations. The maximum and mean depths are presented in Table 2.

Table 2. Max and mean quantifier depths

	Max	Mean
PST	4	0.66
unexpanded *DZFC*	5	1.31
fully expanded *DZFC*	1235	78.68
partially expanded *DZFC*	552	38.54
PST alternating	3	0.63
unexpanded *DZFC* alternating	5	1.18
fully expanded *DZFC* alternating	422	36.19
partially expanded *DZFC* alternating	239	22.16

It has been said that among actually occurring definitions in mathematics texts, the maximum alternating quantifier depth is three. Insofar as *PST* comes close to what actually occurs in textbooks, the maximum alternating depth of 3 tends to confirm this conjecture.

Note that the maximum depth after translating into *DZFC* goes up to 5. This reflects what we saw in Appendix A, where a definition that used no quantifiers in *PST* turned out to require them after translation into *DZFC*.

The maximum depth of 1235 for a fully expanded definition confirms the necessity of using definitions to package information into manageable chunks. Meanwhile, the contrast between the total expansion maximum, and the partial expansion maximum of 552, demonstrates that the lowest, most foundational definitions, lend quite a bit of this complexity.

The ratio $78.68/36.19 \approx 2.17$ of the mean fully expanded depth to the mean fully expanded alternating depth suggests that quantifiers often occur in runs of two, before alternating, when definitions are written in pure set theory. The somewhat lower ratio of $38.54/22.16 \approx 1.74$ for the partially expanded cases indicates the extent to which the lowest-level concepts contribute to this doubling of consecutive quantifiers.

The mean depth for *PST* alternating (again, what comes closest to what we ordinarily think of as quantifier depth in textbooks) shows that, while the maximum is three, the most common depths are 0 and 1. The exact number of occurrences are presented in Table 3.

Length data. As was expected, there is rapid blowup in the size of definitions when they are expanded. In collecting our data we set a maximum of $2^{31} - 1$ before we stopped counting, and this maximum was often reached.

In particular, since the development of the real numbers taken from Suppes [14] involves such deep definition trees, any definition mentioning the real numbers will

Table 3. Quantifier depth frequencies in *PST*

	Occurrences	
Depth	*PST*	*PST* alternating
0	178	178
1	118	120
2	30	35
3	14	8
4	1	0

have enormous expanded length. For example, the definition of the basis for the standard topology on the reals (see Section 3) is just 303 symbols long after initial translation into *DZFC*, but blows up to over $2^{31} - 1$ symbols after expansion.

The longest definition we formalized from Suppes [14] was 526 symbols, and the longest from Munkres [11] was 714 symbols.

References

1. Aho, A.V. and Ullman, J.D.: *The Theory of Parsing, Translation, and Compiling, volume 1.* Prentice-Hall, Englewood Cliffs, N.J., 1972.
2. Beeson, M.J.: *Foundations of Constructive Mathematics.* Springer, Berlin, 1985.
3. Bertot, Y. and Castéran, P.: *Interactive theorem proving and program development: Coq'Art: The calculus of inductive constructions.* Springer, Berlin, 2004.
4. Feferman, S.: Definedness. *Erkenntnis*, 43(3):295–320, 1995. Varia with a Workshop on the Foundations of Partial Functions and Programming (Irvine, CA, 1995).
5. Friedman, H. and Flagg, R.C.: A framework for measuring the complexity of mathematical concepts. *Adv. in Appl. Math.*, 11(1):1–34, 1990.
6. Friedman, H.: Proofless text. Manuscript, September 29, 2005.
7. Gordon, M.J.C. and Melham, T.F., editors: *Introduction to HOL: A theorem proving environment for higher-order logic.* Cambridge University Press, 1993.
8. Harrison, J.: HOL light: a tutorial introduction. In Mandayam Srivas and Albert Camilleri, editors, *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design*, pages 265–269, 1996.
9. Kieffer, S.: A language for mathematical knowledge management. Master's thesis, Carnegie Mellon University, 2007.
10. Kohlhase, M.: *OMDoc: An open markup format for mathematical documents*, volume 4810 of *LNAI.* Springer, Berlin, 2006.
11. Munkres, J.R.: *Topology.* Prentice Hall, Upper Saddle River, N.J., second edition, 2000.
12. Nipkow, T., Paulson, L., and Wenzel, M.: *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science.* Springer, Berlin, 2002.
13. Rudnicki, P.: An overview of the Mizar project. In *1992 Workshop on Types for Proofs and Programs.* Chalmers University of Technology, Bastad, 1992.
14. Suppes, P.: *Axiomatic Set Theory.* Van Nostrand, Princeton, 1960.
15. Troelstra, A.S. and Schwichtenberg, H.: *Basic Proof Theory.* Cambridge University Press, Cambridge, second edition, 2000.

16. Wenzel, M.: Isabelle/Isar – a generic framework for human-readable proof documents. *Studies in Logic, Grammar, and Rhetoric*, 10(23), 2007. From Insight to Proof – Festschrift in Honour of Andrzej Trybulec, edited by R. Matuszewski and A. Zalewska.

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 18 (31) 2009

How to Define Terms in Mizar Effectively

Artur Korniłowicz

Institute of Computer Science
University of Białystok, Poland
artur@math.uwb.edu.pl

Abstract. This paper explains how proofs written in Mizar can evolve if some dedicated mechanisms for defining terms are used properly, and how to write articles to fully exploit the potential of these mechanisms. In particular, demonstrated examples show how automatic expansion of terms and terms identification allow to write compact, yet readable proofs.

1 Introduction

It is commonplace that new authors writing their first Mizar [1] articles learn how to use the language and the verification system by looking at basic articles stored in the Mizar Mathematical Library. This is in principle the correct way, but the problem is that some of these articles were written many years ago, when many features currently available in Mizar had not been implemented yet. The Library Committee keeps revising the articles quite successively, rewriting proofs to use new features, but there is still much work to be done in this area. In fact it is probably a never ending process, since stronger and stronger mechanisms are still being added to the verifier to increase reasoning automation and make proofs much shorter, sometimes almost trivial.

We will take some articles written at the beginning of the previous decade as our working examples. We will focus particularly on automatic expansions of terms defined with the `equals` (this feature was implemented in August 2005 in Mizar version 7.6.01) and the identification of terms using `identify` (implemented in August 2006 in version 7.8.01).

2 Definitions

Mizar is an open language, i.e. users can introduce new symbols and define new notions. Symbols are qualified with the kinds of notions that can be defined, i.e. predicates, attributes, modes, functors, structures, selectors and left and right functorial brackets.

In this section we will focus on modes and functors only. In particular, we will give some hints on how their result types should be defined effectively.

Let us look at the following example, taken from [10].

ISBN 978-83-7431-229-5 ISSN 0860–150X

```
definition
  let A, B be set;
  mode FUNCTION_DOMAIN of A,B -> functional non empty set means
:: FRAENKEL:def 2
  for x being Element of it holds x is Function of A,B;
end;
```

It defines a `functional non empty set`, named `FUNCTION_DOMAIN`, consisting of functions from any set `A` into a set `B`, where `functional` means that its every element is a function. Although at a first glance everything may look fine, we will analyze the result type of this definition in more detail.

The author's intention was to ensure that `FUNCTION_DOMAIN` is always non-empty. So the adjective `non empty` has been stated. This is correct. But the property `functional` can be proven based on the definiens, and therefore should not be a part of the result type. Of course, we would like to know that every `FUNCTION_DOMAIN` is `functional`. The best way to achieve that is to make a conditional registration:

```
registration
  let A, B be set;
  cluster -> functional FUNCTION_DOMAIN of A,B;
end;
```

Then the definition may be changed to:

```
definition
  let A, B be set;
  mode FUNCTION_DOMAIN of A,B -> non empty set means
  for x being Element of it holds x is Function of A,B;
end;
```

Why is this version accompanied with the registration better than the original definition? The point is seen when one introduces a functor with the result type `FUNCTION_DOMAIN`, like, for example, in [7]

```
definition
  let UA be Universal_Algebra;
  func UAAut UA ->
       FUNCTION_DOMAIN of the carrier of UA, the carrier of UA
  means
:: AUTALG_1:def 1
  for h being Function of UA, UA holds
      h in it iff h is_isomorphism UA, UA;
end;
```

To properly define functors, the Mizar verifier requires proving two correctness conditions: `existence` saying that there exists an object of the type `FUNCTION_DOMAIN` satisfying the definiens, and `uniqueness` saying that there exists exactly one such

object. With the first condition in mind, it is obviously easier to construct an object with a less complex mother type, the "weaker" `FUNCTION_DOMAIN` is more convenient, and `non empty set` is obviously "weaker" than `functional non empty set`. One disadvantage of this approach is that the extra conditional registration must always be imported to fully exploit the definition. But once we have the registration imported the checker knows that not only `UAAut`, but all other functors with the result type `FUNCTION_DOMAIN` are `functorial` without any proof. And this is the real gain in the end.

A slightly different situation is when one introduces a new functor. The result type of a functor must contain the information needed to formulate its definiens in a natural and concise way and to allow proving its `uniqueness`. For example, in

```
definition
  let f1,f2 be complex-valued Function;
  func f1 + f2 -> Function means
:: VALUED_1:def 1
  dom it = dom f1 /\ dom f2 &
  for c being set st c in dom it holds it.c = f1.c + f2.c;
end;
```

`f1+f2` must be of the type at least `Function` since the application operator (`.`), which is defined for functions, is applied to it.

A typical example showing that some adjectives are required in the mother type of a functor is a definition of a structural object, where needed selectors are described. For example, in the definition of the product of two relational structures (see [6])

```
definition
  let X, Y be RelStr;
  func [:X,Y:] -> strict RelStr means
:: YELLOW_3:def 2
  the carrier of it =
      [:the carrier of X, the carrier of Y:] &
  the InternalRel of it =
      ["the InternalRel of X, the InternalRel of Y"];
end;
```

the adjective `strict`, saying that there are no other fields in the defined structure, is necessary to prove the `uniqueness` condition. `RelStr` is a relational structure which can be a predecessor of, for example, a relation structure extended by a topology. So, it is clear that not all relational structures with the carrier and the internal relation described above are equal. But if the structures are both `strict`, they are equal.

3 Theorems and Registrations

In this section we demonstrate what theorems and registrations should be stated in an article to achieve short proofs and use the full Mizar power.

As an example let us take a proof of the associativity of the supremum in the lattice of natural numbers with the GCD and LCM operations taken from [4].

A very naive proof which does not use too many MIZAR features would look like:

```
registration ::R1
  cluster Nat_Lattice -> join-associative;
  coherence
  proof
    let p, q, r be Element of Nat_Lattice;
    set L = the L_join of Nat_Lattice;
    set o = lcmlat;
a1: L = o by Def5;
    reconsider p1 = p, q1 = q, r1 = r as Element of NAT by Def5;
    thus p"\/"q"\/"r = L.(p"\/"q,r) by LATTICES:def 1
    .= L.(L.(p,q),r) by LATTICES:def 1
    .= o.(p1 lcm q1,r) by a1,Def4
    .= p1 lcm q1 lcm r1 by Def4
    .= p1 lcm (q1 lcm r1) by NEWTON:56
    .= o.(p,q1 lcm r1) by Def4
    .= L.(p,L.(q,r)) by a1,Def4
    .= L.(p,q"\/"r) by LATTICES:def 1
    .= p"\/"(q"\/"r) by LATTICES:def 1;
  end;
end;
```

where `Def4` is a definition of the operation playing role of the supremum and `Def5` is a definition of the lattice, both taken from [4].

The first step to make the proof shorter is to introduce a theorem showing the correspondence between the operation in the lattice and the operation on numbers, like:

```
theorem T0:
  for x, y being Element of Nat_Lattice
  for m, n being Nat st x = m & y = n holds
  x "\/" y = m lcm n
  proof
    let p, q be Element of Nat_Lattice;
    let p1, q1 be Nat such that
a1: p = p1 & q = q1;
    thus p"\/"q = (the L_join of Nat_Lattice).(p,q)
                  by LATTICES:def 1
    .= lcmlat.(p,q) by Def5
    .= p1 lcm q1 by a1,Def4;
  end;
```

which gives the more concise proof of the registration

```
registration ::R2
  cluster Nat_Lattice -> join-associative;
  coherence
  proof
    let p, q, r be Element of Nat_Lattice;
    reconsider p1 = p, q1 = q, r1 = r as Element of NAT by Def5;
a1: q"\/"r = q1 lcm r1 by T0;
    p"\/"q = p1 lcm q1 by T0;
    hence p"\/"q"\/"r = p1 lcm q1 lcm r1 by T0
    .= p1 lcm (q1 lcm r1) by NEWTON:56
    .= p"\/"(q"\/"r) by a1,T0;
  end;
end;
```

Observe that because the operator `lcm` can only be applied to numbers, extra variables `p1`, `q1` and `r1` are needed inside the proof to switch from the lattice context to numbers, and vice versa. In such a situation the conditional registration

```
registration
  cluster -> natural Element of Nat_Lattice;
  coherence;
end;
```

helps to make it even simpler. It allows to treat elements of the lattice as natural numbers, and then the theorem `T0` can be reformulated as:

```
theorem T1:
  for x, y being Element of Nat_Lattice holds x "\/" y = x lcm y
  proof
    let p, q be Element of Nat_Lattice;
    thus p"\/"q = (the L_join of Nat_Lattice).(p,q)
                  by LATTICES:def 1
    .= lcmlat.(p,q) by Def5
    .= p lcm q by Def4;
  end;
```

and the registration `R2` as:

```
registration ::R3
  cluster Nat_Lattice -> join-associative;
  coherence
  proof
    let p, q, r be Element of Nat_Lattice;
a1: q"\/"r = q lcm r by T1;
    p"\/"q = p lcm q by T1;
    hence p"\/"q"\/"r = p lcm q lcm r by T1
    .= p lcm (q lcm r) by NEWTON:56
    .= p"\/"(q"\/"r) by a1,T1;
```

```
  end;
end;
```

In this case less variables are needed, which clearly makes the proof easier.

Another feature which increases the deduction power of the MIZAR checker is automatic expansion of terms defined using the `equals` keyword. In the above example, several times we explicitly referred to the definition:

```
definition
  let G be non empty \/-SemiLattStr, p, q be Element of G;
  func p "\/" q -> Element of G equals
:: LATTICES:def 1
  (the L_join of G).(p,q);
end;
```

However, this is not really necessary. It is enough to extend the article's environment by the directive `definitions LATTICES`, where the notion was defined, see [12]. With this directive in effect, the checker will always automatically equate all occurrences of `"\/"` with `the L_join`. That process makes the theorem `T1` almost obvious:

```
theorem
  for x, y being Element of Nat_Lattice holds
  x "\/" y = x lcm y by Def4;
```

please note that even if the proof is trivial, the theorem should be stated and referred to when needed. But one of the newest enhancements of the MIZAR checker makes even such references unnecessary by identifying selected terms internally. This feature is discussed in the next section.

4 Terms Identification

In mathematical practice, a given object or an operation is often treated in many different ways depending on contexts in which they occur. A natural number can be considered as a number, or as a finite ordinal. The least common multiply can be considered as an operation on numbers, or as the supremum of elements of some lattice, as we do in our example. In such cases it is often worthwhile to have 'translation' theorems, like the last one in the previous section. But it would be really comfortable for the users, if they did not have to refer to such theorems, but rather had them 'built-in' some way. The developers of MIZAR decided to implement such a feature, naming it *terms identification*, and introducing a new keyword, `identify`, in the MIZAR language. Its syntax is the following:

```
Identify-Registration =
"identify" Functor-Pattern "with" Functor-Pattern
[ "when" Variable-Identifier "=" Variable-Identifier
   { "," Variable-Identifier "=" Variable-Identifier } ] ";"
Correctness-Conditions .
```

where `Identify-Registration` is a part of the rule

```
Registration-Block = "registration"
{ Loci-Declaration | Cluster-Registration | Identify-Registration }
"end" .
```

and `Correctness-Conditions` is `compatibility` described later.

The aim of the identification is matching the term at the left side of the `with` keyword with the term stated at the right side, whenever they occur together. The current implementation (version `7.11.01`) allows matching in one direction only, i.e. when the verifier processes a sentence containing the left side term, it generates its local copy with the left side term symbol substituted by the right side one and makes both terms equal to each other. Such a equality allows to justify facts about the left side terms via lemmas written about the right side ones, but not vice versa. In this sense identification is not symmetric, which is showed with the following example. First, we introduce the registration:

```
registration
  let p, q be Element of Nat_Lattice;
  identify p "\/" q with p lcm q;
  compatibility;
```

The lemma

```
L1: for x, y, z being Element of Nat_Lattice holds
    x lcm y lcm z = x lcm (y lcm z);
```

can be used to justify the sentence

```
L2: for x, y, z being Element of Nat_Lattice holds
    x "\/" y "\/" z = x "\/" (y "\/" z) by L1;
```

but justifying `L1` with `L2` directly does not work.

Let us now see at the proof of the associativity of the supremum that uses this mechanism:

```
registration
  cluster Nat_Lattice -> join-associative;
  coherence
  proof
    let p, q, r be Element of Nat_Lattice;
    thus thesis by NEWTON:56;
  end;
end;
```

Comparing it with `R3`, the power of terms identification becomes evident.

4.1 Some Technical Aspects

Correctness Conditions Terms identification can be used to identify two terms built with different functor symbols, but also with the same symbol. In the case when different variables are used at both sides of `with`, a `when` clause must be used to establish the correspondence between appropriate arguments. Depending on whether the `when` clause occurs or not, the system generates two different conditions:

```
registration
  let p, q be Element of Nat_Lattice;
  let m, n be Nat;
  identify p "\/" q with m lcm n when p = m, q = n;
  compatibility
  proof
    thus p = m & q = n implies p "\/" q = m lcm n;
  end;
end;

registration
  let p, q be Element of Nat_Lattice;
  identify p "\/" q with p lcm q;
  compatibility
  proof
    thus p "\/" q = p lcm q;
  end;
```

Identification Visibility Terms identification is available immediately at the place where it is introduced till the end of the article. If one wants to use the identification introduced in an external article, it should be imported in the environment. The current implementation of identification is internally similar to registrations, so identification does not have a library directive on its own, so identifications are imported with `registrations`.

4.2 Typical Errors Reported

```
registration
  let p, q be Element of Nat_Lattice;
  identify p "\/" q with 1_NN;
::>                          *189,189
end;

::> 189: Left and right pattern must have
         the same number of arguments
```

In this case the error description offered by the checker is self-explanatory.

```
registration
  let p, q be Element of Nat_Lattice;
  let m, n be Nat;
  identify p "\/" q with m lcm n when p = m, q = n;
::>                                      *139   *139
end;

::> 139: Invalid type of an argument.
```

This error means that the types of variables `p` and `q` do not round up to the types of `m` and `n`, respectively. The solution to the problem is the registration:

```
registration
  cluster -> natural Element of Nat_Lattice;
  coherence;
end;
```

4.3 Examples of Use

Here we list some natural and useful identifications introduced in the Mizar Mathematical Library.

```
registration
  let a, b be Element of G_Real, x,y be real number;
  identify a+b with x+y when a = x, b = y;
end;

registration
  let a be Element of G_Real, x be real number;
  identify -a with -x when a = x;
end;
```

registered in [8], where `G_Real` is the additive group of real numbers.

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

registered in [5], where `Real_Lattice` is the lattice of real numbers with the `max` and `min` operations.

```
registration
  let a, b be Element of INT.Group;
  identify a*b with a+b;
end;
```

introduced in [9], where `INT.Group` is the additive group of integers.

`identify` can be used not only when structural objects are constructed, but also in, so called, classical cases:

```
registration
  let X, D be non empty set,
      p be Function of X,D, i be Element of X;
  identify p/.i with p.i;
end;
```

registered in [2], where `p/.i` is a restricted application, defined in [3].

```
registration
  let p be XFinSequence;
  identify len p with dom p;
end;
```

introduced in [11], where `len` stands for the length and `dom` for the domain of a finite sequence.

```
registration
  let x, y be real number, a, b be complex number;
  identify x+y with a+b when x = a, y = b;
  identify x*y with a*b when x = a, y = b;
end;
```

defined in XXREAL_3, where `x+y` and `x*y` are operations on extended reals and `a+b` and `a*b` are defined for complex numbers.

5 Conclusions

When a new feature is implemented in a system coupled with a database of source files, like Mizar and the Mizar Mathematical Library, it is desirable to 're-write' the database to exploit new possibilities. But in general it is not possible to find all contexts where these new features could be used, e.g. it is an undecidable problem whether two terms are equal or not, and then no automatic tools can be invented to find all cases where terms identification could be registered. Therefore, on behalf of the Mizar Library Committee, with this paper we would like to issue an appeal to Mizar users to get more involved in the continuous process of Mizar Mathematical Library revisions motivated either by finding better ways of formalization of some facts, or by implementation of stronger mechanisms in the checker. The simplest thing that all users could do is to report what useful revisions can, or should, be processed.

We showed in this paper that terms identifications definitely make proofs more compact: the original proof of associativity of the supremum was 14 lines long, while the new one using automatic expansion of terms and terms identification had only 2 lines. So it would be valuable to gather from all Mizar users more information on what terms identifications of notions already defined in the Mizar Mathematical Library could be registered.

References

1. Mizar homepage: `http://mizar.org`.
2. Czesław Byliński. Functions from a Set to a Set. *Formalized Mathematics*, 1(**1**):153–164, 1990. MML Id: `FUNCT_2`.
3. Czesław Byliński. Partial Functions. *Formalized Mathematics*, 1(**2**):357–367, 1990. MML Id: `PARTFUN1`.
4. Marek Chmur. The Lattice of Natural Numbers and The Sublattice of it. The Set of Prime Numbers. *Formalized Mathematics*, 2(**4**):453–459, 1991. MML Id: `NAT_LAT`.
5. Marek Chmur. The Lattice of Real Numbers. The Lattice of Real Functions. *Formalized Mathematics*, 1(**4**):681–684, 1990. MML Id: `REAL_LAT`.
6. Artur Korniłowicz. Cartesian Products of Relations and Relational Structures. *Formalized Mathematics*, 6(**1**):145–152, 1997. MML Id: `YELLOW_3`.
7. Artur Korniłowicz. On the Group of Automorphisms of Universal Algebra & Many Sorted Algebra. *Formalized Mathematics*, 5(**2**):221–226, 1996. MML Id: `AUTALG_1`.
8. Eugeniusz Kusak, Wojciech Leończuk, and Michał Muzalewski. Abelian Groups, Fields and Vector Spaces. *Formalized Mathematics*, 1(**2**):335–342, 1990. MML Id: `VECTSP_1`.
9. Dariusz Surowik. Cyclic Groups and Some of Their Properties – Part I. *Formalized Mathematics*, 2(**5**):623–627, 1991. MML Id: `GR_CY_1`.
10. Andrzej Trybulec. Function Domains and Frænkel Operator. *Formalized Mathematics*, 1(**3**):495–500, 1990. MML Id: `FRAENKEL`.
11. Tetsuya Tsunetou, Grzegorz Bancerek, and Yatsuka Nakamura. Zero-Based Finite Sequences. *Formalized Mathematics*, 9(**4**):825–829, 2001. MML Id: `AFINSQ_1`.
12. Stanisław Żukowski. Introduction to Lattice Theory. *Formalized Mathematics*, 1(**1**):215–222, 1990. MML Id: `LATTICES`.

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 18 (31) 2009

The Influence of Delocalization on the Results of Eliminating Repetitions of Semantically Equivalent Sentences in the MML Database

Robert Milewski

Institute of Computer Science
University of Białystok
Sosnowa 64, Białystok, Poland,
milewski@math.uwb.edu.pl

Abstract. Detecting and removing repetitions of semantically equivalent sentences improves the quality of the MML database, but it is also an important factor in evaluating the robustness of the Mizar system. However, the complicated structure of proofs in the MML database makes it very difficult to automatically discover and remove such repetitions. One possible solution to this problem may be to apply the delocalization process, i.e. moving some sentences to outer levels of a proof where the elimination of repetitions becomes possible.

1 Main Ideas

The Mizar Mathematical Library (MML) is continuously developed by users of the Mizar system [5], but it also undergoes frequent revisions. The necessity of these changes is connected with the need of preserving the high quality of the database [7]. On the other hand, extensive modifications allow to detect cases of atypical system behavior. Similarly to testing monotonicity or permutability of references [3], such revisions may be used to analyze the robustness and increase the quality of the Mizar system.

In this paper we discuss detecting and eliminating repetitions of semantically equivalent sentences in the MML. Preliminary experiments with this issue have already shown some positive results. However, the results were not satisfactory, because a lot of repetitions found in the process could not be removed automatically. Then it became evident that the process of delocalization (moving sentences to the highest possible level of the proof structure) can solve this problem. In the sequel we present results of two experiments that have been carried out: first without, and then with delocalization.

2 Detecting and Removing Repetitions of Semantically Equivalent Sentences

It is quite common that during writing Mizar articles authors use the same sentence many times. It also happens that these sentences are justified several times

ISBN 978-83-7431-229-5 ISSN 0860–150X

(usually by oversight). Such repetitions unnecessarily increase the length of the formal text and decrease its clarity. In the graph of references the nodes become duplicated and superfluous edges have to be added. Obviously, this is rather incompatible with the main principles of formalization where a proved statement (once justified with the help of axioms or previously proved theorems) should be regarded as true, and used in all relevant situations. To clear out the library, a specific utility has been implemented which finds semantically equivalent sentences and removes all repetitions.

The utility is called REMEQTH, an acronym for REMoving of EQuivalent Theorems (of course it removes only repetitions of equivalent statements, not all of them). This program analyses sentences within a MIZAR text and checks if there exist equivalent statements on the list of currently available sentences. If there are such cases, their position is stored, so that after checking all the text only the first appearance is retained and all repetitions could be removed. Here the equivalence of statements means that sentences are not syntactically, but semantically equivalent. Therefore REMEQTH cannot be based only on the information generated by the MIZAR parser (unlike the utilities used e.g. for testing permutability or monotonicity of references [3]). REMEQTH also needs some information from the more involved MIZAR pass, the PREPARATOR, in which the logical form of sentences is available. All sentences at that point are transformed into the form of semantic correlates, i.e. they are built with the general quantifier, conjunction and negation only. Only comparing sentences in this form can give the intended results, since only in this form two semantically equivalent sentences are the same. Restricting to the information generated by the parser, it would not be possible to state the equivalence of sentences like below:

```
A implies B;

not (A and not B);
```

In this case only the statements equivalent with respect to the relation r_0 (as defined in [4]) would give any search results. Practically speaking, most of repeated sentences would not be found.

If there exist more than two equivalent sentences, then of course all possible pairs should be found, i.e. $\binom{n}{2}$ pairs for n equivalent sentences. In practice, REMEQTH changes all references to repetitions of semantically equivalent sentences into references to the first occurrence. Then it is enough to apply the utility which removes all unused labels (CHKLAB) to remove all labels of the repetitions of semantically equivalent sentences (since no reference can refer to them anymore), and finally call another tool to remove all irrelevant and not labeled sentences (INACC). As a result all repetitions are removed.

The REMEQTH utility changes just the references, so considering only single statements, its level of alteration to the original text is on the level of the r_3 relation [4]. However, elimination of repetitions of equivalent sentences in the whole text changes the structure of proofs, and so the equivalence of texts can be true only on the level of the r_4 relation.

Let us note that REMEQTH does not remove equivalent sentences located in different sub-blocks, i.e. when the first sentence is not available on the level of the other one. It is possible, howewer, and that situation is quite frequent in the MML database, that there are many sub-blocks in the proof containing equivalent sentences:

```
proof
  ..........
  proof
    sentence1;
    ..........
  end;
  ..........
  proof
    sentence1;
    ..........
  end;
  ..........
end;
```

Both occurrences of `sentence1` are equivalent and one of them should be removed, but it cannot be done with the procedure described above. In such cases we can use delocalization which moves all sentences (that can be moved) from subproofs to the highest possible level of the proof structure. This process is described in next sections.

There is a specific context where equivalent sentences are found by REMEQTH, but the repetitions cannot be removed. It happens when the second equivalent sentence is a part of the thesis in the current proof:

```
A1: sentence1;
sentence2 by A1;
..........
hence sentence1 by A1;
```

The tactic used by REMEQTH assumes that every reference pointing to the repetition of some sentence is changed into a reference to the first equivalent sentence, and next redundant labels and unused fragments of the text are removed. In the above case, this procedure does not result in removing the repetition. The second sentence is a part of the thesis in the current proof, so it cannot be recognized as redundant, and so the INACC tool does not remove it.

3 The Results of Eliminating Repetitions of Semantically Equivalent Sentences

The REMEQTH tool was used to carry out the elimination of repetitions of semantically equivalent sentences in the whole MML database. The main steps of this experiment are described below (cf. [1], [3]):

- processing the MML with the FORMATER utility and storing the formatted texts for future comparison,
- processing the MML with DELINKER and SEPREF tools,
- replacing references to repetitions with references to the first equivalent sentence (REMEQTH),
- removing redundant labels and unnecessary parts of texts,
- reverting the changes made by DELINKER and SEPREF (LINKER, CHKLAB, TOHEREBY, RENTHLAB, SORTREF),
- final formatting with FORMATER.

The formatting is necessary for standardizing the texts. It helps to avoid differences resulting from unintentional change of the text. Comparing the size of articles after first and last formatting, we obtained the following results: the size of the whole MML library was decreased from 51 275 019 bytes to 51 038 106 bytes. It means saving 236 913 bytes, i.e. about 0.46 % of the initial size.

There were 8101 pairs of equivalent sentences found in this experiment. Of course, it does not mean that 8101 sentences were removed. Taking into account that it is a number of all possible pair combinations, and the impossibility of removing a sentence if it is the part of a thesis, the actual number of removed sentences is about half of it. There were 4807 replaced references and 4355 removed sentences (the total number of sentences decreased from 1 339 485 to 1 335 130). Because the texts were formatted, the number of removed sentences was equal to the number of saved text lines (since after formatting one sentence is always one line of text).

Apart from cleaning the MML, the graph of references was also simplified: there were more than 4000 nodes removed together with relevant edges.

4 Delocalization

As mentioned previously, REMEQTH cannot compare sentences in different proof sub-blocks. To solve this problem we should move to outer blocks all sentences which do not depend on constants and variables introduced in the current block, and are not justified by references to other sentences within the current block. This process is called delocalization and is performed by the DELOCAL utility. Of course there are situations when we have to move a sentence more then one level up. This could be handled in two ways: create a rather complicated procedure to determine on which level to put the sentence, or simply move a sentence one level up and apply this process repeatedly as long as there is nothing more to move. DELOCAL uses the second approach, which is much simpler, but is more time-consuming.

It may not be obvious to say whether the delocalization improves or spoils Mizar texts. But from the formal point of view, it does improve them and make them logically "cleaner". It is hard to justify why to place in a sub-block sentences which do not depend on constants and variables introduced in this sub-block, and also do not refer to other sentences within that block. Such sentences should rather be placed as high as possible in the structure of sub-blocks of the proof. This

approach seems natural, gives the possibility to refer to those sentences within a bigger part of the proof.

The delocalization has a huge influence on the process of detecting semantically equivalent sentences. Obviously, when applied to delocalized texts, the REMEQTH utility is able to find all equivalent sentences that depend on the same constants and variables (two sentences that depend on different constants or variables are not considered equivalent).

One may wonder why authors of Mizar articles so frequently place certain sentences so deeply in the structure of their proofs when it is not necessary and the same sentence placed on the higher level would be accessible within a bigger part of the proof. One of the reasons may be some kind of laziness. Often while proving there appears a need to add one more new premise and refer to it, and the simplest solution may seem to write that premise directly before the current sentence and to refer to it with `then` without introducing any new labels. Placing that premise e.g. a dozen or so lines before (on a higher proof level and between other sentences concerning completely different subject matters) may seem unnatural and can negatively affect the readability of the proof. This is the main reason why the REMEQTH utility has not been applied as yet to permanently change the articles stored in the MML, although it would have been a significant simplification of the structure of the library. Still, REMEQTH can be useful for experiments concerning data-mining and the robustness of the Mizar system.

Before the delocalization the MML must be prepared with DELINKER [3] which removes references to the preceding sentences (`then` and `hence` linking), but also standardizes the names of labels. In fact the standardization is a side effect of DELINKER, but it makes easier the process of removing repetitions of semantically equivalent sentences. After that we can be certain that any two different statements have different labels. It allows to avoid situations when the meaning of a reference is changed after moving a sentence to a higher level of the proof.

In practice, in order to check if it is possible to move a sentence to a higher level, the utility writes down the amount of defined labels at the moment of opening another block of text. Because of using DELINKER, this amount is equal to the number stored in the name of the recently defined label. Analyzing a given sentence we examine all references in its justification. The utility checks if the referred sentences are in the current block, or before. If all referred sentences are before the beginning of the current block, it is valid to move the sentence to the higher level in the structure of the proof, as far as the references are concerned.

The other condition which has to be checked is whether there are in the sentence any constants defined in the current block. The constants may be introduced with `set`, `take`, `let`, `consider` and `reconsider`. The utility writes down the number of such objects at the moment of opening another block of text, and next checks the numbers of constants that occur in the analyzed sentence. Finally, if all numbers are less than or equal to the one stored earlier, and the previous condition concerning labels gives a positive result, the curent sentence can safely be moved to a higher level. Its new location is set directly before the beginning of the current block.

As mentioned above, sentences using constants introduced by some language constructs cannot be moved to a higher level. If it is the only reason which makes

the delocalization of the sentence impossible, we may also consider moving the relevant constructs (mainly `set`, `consider` and `reconsider`, because `take` and `let` are a part of the proof skeleton and moving them to the higher level of the proof would destroy the proof structure). To delocalize them we must be sure that moving them to a higher level would not override some other constant with the same name. Such a situation is presented below:

```
consider x being set;
now
  ..........
  then A1: P(x);
  consider x being set such that A2: Q(x);
  A3: R(x) by A2;
  ..........
end;
```

Let us assume that the sentence labeled `A3` does not depend on labels and constants defined in the current `now-end` block (of course beyond x). Then we would want to move this sentence to the higher level - directly before the `now` block. It is only the definition of x that does not allow doing so, but this definition does not depend on labels and constants from the current block, therefore this definition can be moved too. Thus the example after delocalization would look as follows:

```
consider x be set;
consider x be set such that A2: Q(x);
A3: R(x) by A2;
now
  ..........
  then A1: P(x);
  ..........
end;
```

But in this text the sentence labeled `A1` depends now on the x constant which is defined by the second consider (moved before the now-end block) rather than the first one, as was formerly the case. It means that the meaning of this sentence has changed, and such a situation is unacceptable.

To be sure that such a situation does not happen, we would have to change the names of all constants, and give them unambiguous names according to some fixed scheme. But planning such an algorithm we come across a problem connected with reservations of variables. Let us consider the following situation:

```
reserve x for set;
reserve y for Subset of x;

  let x;
  ..........
  consider y;
```

Proceeding with changing the names of variables, the x variable would get a specific name at the moment it is introduced. But the y variable which is introduced later (with `consider`), depends on the variable whose name is used in the reservation block, and which is different from the new name of the variable created with `let`. In this way at the moment of introducing the y variable a new extra variable of the type set is created that y depends on, and it is not the constant introduced by `let`. This situation shows that to create a tool which would automatically change names of constants, it is necessary to remove first all reservations and supply the types of variables at the point of their introduction.

In the current implementation of the Mizar system reservations are identified at the moment of declaration in the reservation block, not at the moment of introducing a reserved variable. It can lead to the following situation:

```
definition
 let x be set;
 mode Subset of x is Element of bool x;
end;

reserve x for set;
reserve y for Subset of x;

definition
 let x be set;
 redefine mode Subset of x -> ...;
end;

for y holds P[y];
```

The variable y is reserved as Subset of x in terms of the first definition of the `Subset` mode. Next there is a redefinition of `Subset`, but in the sentence:

```
for y holds P[y];
```

the variable y is understood as `Subset` of x in the sense of the original definition, because the type of reserved variable was identified at the moment of reservation. If we tried to add types to all variables at the moment of their introduction and remove reservation blocks, we would have the following sentence:

```
for y be Subset of x holds P[y];
```

with the redefined `Subset`. Such a situation is of course unacceptable. In consequence, creating an automatic tool which would eliminate reservations is not possible in the current implementation of the Mizar system. If we tried to change the implementation of the Mizar system to have dynamic interpretation of variables types at the moment of their introduction, it would cause other problems: it would be impossible to use the original definition after a redefinition which in the current implementation is realized by reservations of variables before a redefinition, and it allows to use variables with a reserved, not redefined type.

Therefore in the current implementation of the MIZAR system the delocalization of definitions of constants is just impossible.

The DELOCAL tool changes the form of proofs significantly. When we look at the MML as a graph of references, we see that DELOCAL influences the number and location of nodes and also edges. But it does not change the abstracts of created texts, so using the previously mentioned classification, DELOCAL interferes in the original text on the fourth level.

5 The Results of Delocalization

Below we present some statistics concerning the delocalization applied to the whole MML. DELOCAL was executed 2983 times, i.e. on average about 3.82 times per article. The biggest number of delocalizations for one article is 29 - in the article JGRAPH_2 [6]. The program has to be run several times when a sentence is moved several levels up, but also when there exists a whole block of sentences among which the next one referrs to the previous one. Such a block would not be moved at a time, but rather each run would move only the first sentence, because the others depend on it. This situation is illustrated by the following example:

```
proof
  A1: sentence1;
  A2: sentence2 by A1;
  A3: sentence3 by A2;
  A4: sentence4 by A3;
  A5: sentance5 by A4;
  thus thesis by A5;
end;
```

With the text as above DELOCAL would have to be run five times. At the beginning it would move `sentence1`, then `sentence2` and so on. In the `proof-end` block only the thesis would remain. The thesis of course cannot be moved, because it is a part of the proof skeleton, but in such case we can think of eliminating a proof which consists of the conclusion only.

The difference in article sizes is not interesting in this case, because the utility moves sentences within the same article (only changes the order of sentences). Therefore possible differences in size are caused by the formatting (connected with the equivalence on the first level).

Running the DELOCAL tool on the whole MML resulted in moving 68787 sentences, i.e. on average about 23.06 times for one run, and about 88.08 times for one article.

6 The Results of Eliminating Repetitions of Semantically Equivalent Sentences after Delocalization

Below we present the procedure of eliminating repetitions of semantically equivalent sentences after the delocalization process, and compare the results with these showed in Section 3 (without delocalization).

The experiment was carried out according to the following scheme:

- the first formatting of the MML to fix the format of texts,
- processing the MML with DELINKER and SEPREF,
- first elimination of repetitions of semantically equivalent sentences,
- restoring the format of texts changed by DELINKER and SEPREF with auxiliary utilities [2],
- second formatting of the MML to compare the current state with the original texts,
- preparing the MML for the delocalization with auxiliary software,
- delocalization,
- processing with DELINKER and SEPREF,
- second elimination of repetitions of semantically equivalent sentences,
- restoring the format of texts changed by DELINKER and SEPREF,
- third formatting of the MML to compare the current state with the previous one.

The above experiment showed a big influence of delocalization on the number of detected and removed repetitions of semantically equivalent sentences. When we compare the size of the formatted MML at the beginning (state 1), after the first elimination of repetitions of semantically equivalent sentences (state 2), and at the end (state 3), we obtain the following results:

State 1	55 403 103
State 2	55 221 676
Difference 1	181 427
Difference 1 (%)	0.33 %
State 3	54 817 606
Difference 2	404 070
Difference 2 (%)	0.73 %
Total difference	585 497
Total difference (%)	1.06 %

The number of removed repetitions in both elimination procedures is shown in the following table:

First elimination	4 029
Second elimination	9 200
Total	13 229

As far as the graph of references is concerned, the elimination of repetitions of semantically equivalent sentences preceded by the process of delocalization decreased the number of nodes by 13299. Each removed node had its semantically equivalent counterpart, so all edges starting from the removed nodes were moved to their equivalent counterparts. The large number of removed nodes allows to claim that the described procedure may be used to significantly improve the quality of the MML.

7 Final Remarks

Although the delocalization changes only the position of selected sentences in a proof structure, its influence on the results of detecting and eliminating repetitions of semantically equivalent sentences is very significant. The number of repetitions removed after the delocalization is more than three times bigger than without it. Therefore the tentative hypothesis that the delocalization might help to find a big number of semantically equivalent sentences was confirmed. The fact that the size of redundant information removed in the process exceeded one per cent of the initial size of the MML was quite surprising even for the authors of the Mizar system.

8 Acknowledgments

I thank all of those who helped me with writing this paper for their valuable hints and advice. In particular I want to thank A. Trybulec and A. Naumowicz.

References

1. Milewski, R.: Algorithms for Analysis of a System Supporting Formal Deduction, PhD Thesis, Faculty of Computer Science, Bialystok Technical University, 2008.
2. Milewski, R.: New Auxiliary Software for MML Database Management, Mechanized Mathematics and Its Applications, 5(2), pp. 1–10, 2006.
3. Milewski, R.: Robustness of Systems for Formalizing Mathematics – Testing Monotonicity and Permutability of References in Mizar, Mechanized Mathematics and Its Applications, Vol. 4(1), pp. 51–58, 2005.
4. Milewski, R.: Transformations of MML Database's Elements, Lecture Notes in Computer Science, Springer-Verlag, 3863, pp. 376–388, 2006.
5. Mizar Home Page, `http://mizar.org/`.
6. Nakamura, Y.: On Outside Fashoda Meet Theorem, Formalized Mathematics, 9(4), pp. 697–704, 2001.
7. Rudnicki, P.: An Overview of the Mizar Project, Proceedings of the 1992 Workshop on Types for Proofs and Programs, Chalmers University of Technology, Bastad, 1992.

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 18 (31) 2009

Enhanced Processing of Adjectives in Mizar

Adam Naumowicz

Institute of Computer Science
University of Białystok, Poland
adamn@mizar.org

Abstract. As adjectival notions are ubiquitous in informal mathematics, their important role must also be reflected in formal attempts to reconstruct the existing body of mathematical texts. In this paper we describe an enhancement of the Mizar proof checker which enables a more complete automation of notions encoded as adjectives. The proposed improvement concerns the Equalizer – Mizar's module responsible for handling equality, where adjective registrations can be re-used by matching them with classes of equal terms in order to add extra information to inference steps.

1 Introduction

It is common knowledge that successful formalization of mathematical texts requires the underlying formal language be near to the intuition of a mathematician. The fine details of mathematics are much better accounted for in a mixture of mathematical and natural language than in a purely set-theoretical setting. Therefore the languages devised particularly for formalization support the linguistic categories like adjectives that are amply used in informal texts, although they may seem superfluous from the formal point of view. The support for adjectives in the Mizar language dates back to 1983/84, when a version called Mizar HPF (with hidden parameters and functions) was implemented to facilitate a much richer syntax [3]. The idea was also present in de Bruijn's famous Mathematical Vernacular, and is also considered a key feature of its modern derivatives like Wtt [2].

In most (natural) languages that support adjectives, they form an open class of words, i.e. it is relatively common for new adjectives to be formed via derivation. In the formal context, this usually means applying 'technical' suffixes like '-like' or prefixes like 'being_', 'having_', or 'with_' to predicates [3]. When attributes were introduced in Mizar, such changes were done semi-automatically to numerous predicates previously defined in the Mizar library[1].

Since that time, adjectives have been playing a more and more important role helping to minimize information losses in Mizar formalizations. There is still, however, an open area for research on how to improve the processing of adjectival notions in Mizar to imitate even better the use of adjectives in real mathematics. This paper describes an attempt going in this direction.

[1] See also Andrzej Trybulec's posting to the Mizar-Forum mailing list on this topic: http://mizar.uwb.edu.pl/forum/archive/0006/msg00003.html .

ISBN 978-83-7431-229-5 ISSN 0860–150X

2 Registrations of adjective clusters

The paper [6] presents a detailed discussion on how adjectives are handled in MIZAR. Let us just recall here that a "cluster of adjectives" is a collection of attributes (constructors of adjectives) with boolean values associated with them (negated or not) and their arguments. The tree-like hierarchical structure of MIZAR types is built by the widening relation which uses such collections of adjectives to extend existing types [1, 7]. Grouping adjectives in clusters enables automation of some type inference rules. Such rules are encoded in the form of so called registrations. Previously proved registrations can subsequently be used to secure the non-emptiness of MIZAR types (existential registrations), to allow formulating and automating relationships between adjectives (conditional registrations) and to store adjectives that are always true for instantiations of terms with certain arguments (functorial registrations), cf. [6].

Before the proposed enhancement, the role of adjective processing was mostly syntactic, i.e. the Analyzer automatically "rounded-up" the information from all available registrations to disambiguate used constructors and check their applicability. See [5] for a more detailed overview of the internal functioning of the MIZAR `verifier` and the tasks distributed among the Analyzer and the Checker (in particular the Equalizer). The semantic role was restricted to processing only the type information for the terms explicitely stated in an inference. Also, attributive statements as premises or conclusions were not "rounded-up". Neither did the automation take into account the potential of applying registrations to every element of a class of equal terms generated in the Equalizer as a consequence of the equality calculus.

3 Examples

In this section we present some contexts where the automatic type reasoning did not work, while it seemed rather "natural" to expect such statement be accepted by the Checker. The most typical situation concerns attributive statements (`x is <some-attribute>`) or qualifications (`x is <some-type>`) among the premises. Even with the following registration imported from the article `RAT_1`[2]

```
registration
  cluster integer -> rational number;
end;
```

the last line in the listing below was not automatically accepted (the statement was marked with the `::> *4` flag by the Checker):

[2] Unique identifiers, like `RAT_1` in this case, are assigned to all MIZAR articles stored in the MML. The respective files can be found in the `mml` sub-directory of the MIZAR distribution.

```
now
  let a be integer number;
  let b be number;
  a is rational;
  b is integer implies b is rational;
::>                                    *4
end;
```

Although the Checker accepted that `a is rational`, it was not the case for `b`, because the adjective `integer` was only mentioned as a premise and not in the variable declaration, so it was not "rounded-up".

The next example shows an inference where the type of neither constant can be "rounded-up" on its own with a registration taken from `XXREAL_0`, but after processing the equality `a=b` the statement should "naturally" be obvious (because `a` and `b` should share all adjectives as elements of the same equality class):

```
registration
  cluster non negative non zero ->
                        positive (ext-real number);
end;

now
  let a be non negative (real number);
  let b be non zero number;
  a=b implies a is positive;
::>                             *4
end;
```

The enhancement presented in Section 4 makes the Checker accept such statements. Moreover, please note that this more complete "rounding-up" of attributive statements gives also a nice side efect: in the Checker we get something like a contraposition of conditional registrations, because the conclusion is negated when the Checker tries to deduce contradiction, cf. [3].

In this sense registrations are used like theorems, when it is enough to state an implication and then be able to use it in a contraposition form. Of course it does not mean that registering contraposition forms of conditional registrations is not needed anymore – sometimes we want the Analyzer to use them as well (e.g. to match certain notations).

To show an iterative effect of the "rounding-up" done for more complicated terms, we may look at two functorial registrations extracted from the article `ABIAN`:

```
registration
  let i be even Integer, j be Integer;
  cluster i*j -> even;
end;

registration
  let i be even Integer, j be odd Integer;
  cluster i+j -> odd;
end;
```

With these registrations imported, all the following statements are now accepted by the enhanced Checker while the old realization would mark them all as unaccepted:

```
now
   let i,e,o be integer number;
   e is even implies i*e is even;
   e is even & o is odd implies e+o is odd;
   e is even & o is odd implies (i*e)+o is odd;

   let z be complex number;
   z is even Integer implies z*i is even;
end;
```

Let us note that in the above examples all the attributes are absolute, i.e. their only argument is the subject. But in general, the subject may be defined with a type that has its own (explicit or implicit) arguments, and so the adjective has more implicit arguments. For example, with the following definition of continuity as defined in the article `PRE_TOPC`:

```
definition
  let S,T be TopStruct, f be Function of S,T;
  attr f is continuous means
  for P1 being Subset of T st
        P1 is closed holds f" P1 is closed;
end;
```

the function `f` may also map `S` to some space `T1`, and then may not be continuous with respect to `S` and `T1`, so the processing of arguments is crucial to a proper "rounding-up" extension in the Equalizer.

4 The extended "round-up" algorithm

The motivation for this work was presented in [8] showing the limitation of the adjective handling process. As mentioned in Section 2, previously clusters of adjectives in types were "rounded-up" in the Analyzer. The Equalizer just "adjusted" the clusters, trying to move adjectives between various types of the same equality constant if they were applicable. The extended algorithm presented below takes into account the complex structure of terms processed in the Equalizer, that may have numerous representatives, as well as multiple types, which in turn have their arguments of the same form, and so on.

As a class may have several types and several term instances that may match the same registration, the result of matching is a list of instantiations of classes for the loci used in a registration (usually it is just one instantiation, but sometimes there are more, see Section 5 for maximal values noted with current MML).

Technically, the implementation reuses some of the data structures already developed for the Unifier, where an algebra of substitutions is used to contradict a given universal formula, cf. [8]. The main difference is when joining instantiation lists – in the Unifier longer substitution is absorbed, while here the longer substitution remains.

In the calculus of (lists of) instantiations we will use two binary functions, `JOIN` and `MEET` with the following semantics:

- `JOIN(l1,l2)` produces a union of lists `l1` and `l2`, replacing shorter substitutions with longer ones – unlike in the Unifier, where a shorter list is always preferred as it is used for refutation ([8]);
- `MEET(l1,l2)` produces a collection of unions of two instantiations (one from `l1`, the other from `l2` provided they agree on the intersection of their domains; again a shorter substitution is replaced by a longer one if they both are inserted into this collection).

For convenience we also use two lattice-like constants: `TOP` which denotes a trivial substitution (no loci to be substituted, but all constants are matched) and `BOTTOM` which is an empty list of substitutions (no match found). Our `TOP` and `BOTTOM` have the usual lattice properties, e.g. are neutral with respect to the `MEET` and `JOIN` operations, respectively.

Below is an outline of pseudo-code functions that have been implemented to enable matching a class of terms with a given registration. All these functions return as their result a (possibly empty) list of substitutions of classes for loci in the registration. It should be clear which of the `match` functions is used in a certain context looking at the type of their arguments. For simplicity, we treat any class of terms `E` as a special kind of a term – one that satisfies the condition `E is CLASS`.

Let us consider a conditional registration `C`. To check if a given class `E` matches `C` we generate substitutions which match both the type and the antecedent of `C`:

```
match(E:term,C:condreg)
begin
  l:=match(E,C.type)
  l:=MEET(l,match(E,C.antecedent))
  return l
end
```

In the case of a functorial registration `F`, the matching function generates substitutions which match both the registered type and term of `F`. As above, if a substitution is found, it can be used to extend the cluster of the equality class `E`. Let us note that `F.type` is just a radix type, the adjectives from the type's cluster of adjectives do not have arguments other than that of the type, so the cluster does not have to be matched as such:

```
match(E:term,F:funcreg)
begin
  l:=match(E,F.type)
  l:=MEET(l,match(E,F.term))
  return l
end
```

Matching a class `E` with a type `T` is just matching one by one all the types of `E` with `T`:

```
match(E:term,T:type)
begin
  l:=BOTTOM
  if E is CLASS then
    for t in E.types do
      l:=JOIN(l,match(t,T))
  return l
end
```

When types `T1` and `T2` are to be matched, they must denote the same mode (`T1.id=T2.id`) as well as all their arguments must match:

```
match(T1:type,T2:type)
begin
  if T1.id=T2.id then
    begin
      l:=TOP
      while n do
        l:=MEET(l,match(T1.arg(n),T2.arg(n)))
```

```
    end
  else return BOT
  return l
end
```

Matching terms is the main part of the substitution process, since terms are arguments of terms, types and adjectives. Therefore, all matching must eventually come to this point. A class `E` can be matched with a term `T` being a locus in a registration if the type of `T` (`T.type`) and the cluster of adjectives of `T` (`T.cluster`) match the class `E`. Having a valid substitution, we merge it with `(T<-E)` (`E` is substituted for `T`).

If `E` is a class but `T` is not a locus, then we generate a union of possible matches of instances of `E` (taken from `E.terms`) with `T`.

Otherwise, if `E` and `T` have the same kind and number (so `E` is not a class and `T` is not a locus), then we simply match all their arguments:

```
match(E:term,T:term)
begin
  if E is CLASS then
    begin
      if T is LOCUS then
        begin
          l:=match(E,T.type))
          l:=MEET(l,match(E,T.cluster))
          l:=MEET(l,(T<-E))
          return l
        end
      else
        begin
          l:=BOTTOM
          for t in E.terms do l:=JOIN(l,match(t,T))
          return l
        end
    end
  else
    if E.id=T.id then
      begin
        l:=TOP
        while n do
          l:=MEET(l,match(E.arg(n),T.arg(n)))
        return l
      end
    else return BOTTOM
end
```

Matching a class `E` with a cluster of adjectives (for matching an antecedent of a conditional registration or a cluster accompanying the type of a locus) can be split for clarity into the following two steps:

```
match(E:term,L:cluster)
begin
  l:=TOP
  for a in L.adjectives do
    l:=MEET(l,match(E,a))
  return l
end
```

and finally matching single adjectives as below. An adjective `A` matches some adjective in the cluster of a class `E` (`E.cluster`) if they denote the same attribute, have the same boolean value, and their arguments match as well:

```
match(E:term,A:adjective)
begin
  l:=BOT
  if E is CLASS then
    for a in E.cluster do
    if a.id=A.id & a.bool=A.bool then
      begin
        l1:=TOP
        while n do
          l1:=MEET(l1,match(a.arg(n),A.arg(n)))
        l:=JOIN(l,l1)
      end
  return l
end
```

With the matching function as above, we can now present the actual "round-up" algorithm as follows:

0. Create a dependence list for all equivalence classes in a given inference. Let `dep(E)` denote a list of all classes in which `E` appears as a term argument.
1. Put all classes into a set `CLASSES`
2. Proceed as below until `CLASSES` remains empty:

```
while CLASSES <> {} do
  begin
    take E from CLASSES
    repeat
      extended=:false
      for C in CondRegs do
        l:=match(E,C)
        if l<>BOTTOM then
          begin
            extend E.cluster with l
              applied to C.consequent
            extended:=true
          end
      for F in FuncRegs do
        l:=match(E,F)
        if l<>BOTTOM then
          begin
            extend E.cluster with l
              applied to F.consequent
            extended=true
          end
     if extended then
       CLASSES=CLASSES+dep(E)
    until not extended
  end
```

As we see, if the class is matched with some registration, the resulting substitution is used to substitute all loci in the registration's consequent cluster with appropriate classes, this new cluster should be used to extend the cluster of a given class. Please note that "rounding-up" with a conditional registration may enable a functorial one, and the other way round, so the `repeat` loop must guarantee that the algorithm is not sensitive to the order of registrations.

5 Some statistics

Below we present the statistics collected by running main MIZAR utility programs (`verifier`, `relprem`, `trivdemo`, and `relinfer`) equipped with the strengthened Checker as compared to the current official system to demonstrate the performance of the new algorithm. All tests have been performed using MIZAR ver. 7.9.01 and MML ver. 4.103.1019 on a single-processor Intel Xeon 2.8 GHz machine running under the Linux operating system.

Here are the number of calls to the "round-up" procedure for respective utilities:

verifier	860379
relprem	3128080
trivdemo	182984
relinfer	700725

More detailed statistics concerning the equality classes occurring in all inferences were collected for the `verifier`. The biggest number of instantiations generated when "rounding-up" a functorial registration was 17 (in `POLYEQ_3`) and 9 in the case of conditional registrations (in `LIMFUNC2`). The average number of equality classes was 30.366 (max. 828 in `SCMISORT`). The average number of conditional registrations tried was 52.2439 (max. 193 in `JORDAN`) and 163.911 for functorial registrations (max. 671 in `AOFA_I00`). The average number of terms in equality classes was 1.60257. Here we may notice that the biggest number (182) was detected in `FILTER_1` resulting from numerous instances of equal terms built from automatically expanded commutative lattice operations.

The tools `relprem`, `trivdemo` and `relinfer` work with a different population of inferences, so the numbers were slightly different. However, the differences in the case of `relprem` were very small. A noticeable difference was the biggest number of equality classes when running `trivdemo` (728 in `SCMISORT` with a slightly smaller average of 27.7032, and the biggest number of classes for `relinfer`: 826 in `SCMISORT`) with the average of 39.3855 classes.

The extra code to be executed in order to "round-up" the adjectives in the Checker caused a noticeable slowdown in all the utilities. In the table below we present the calculated total running times for the selected utilities applied to the whole library, with the effective times put in brackets (without the extra time needed to load the utilities, accommodate each file, etc.).

Program	Time (effective)	New time	Ratio
verifier	2h 40m (8791s)	4h 48 m (16486s)	1.8753
relprem	16h 43m (59291s)	24h 38m (87812s)	1.4810
trivdemo	2h 44m (9053)	3h 17m (10977s)	1.2125
relinfer	7h 21m (22209s)	9h 27m (29975s)	1.3496

In the following table we compare the times of running the Checker pass only.

Program	Checker	New Checker	Ratio
verifier	5544s	13131s	2.3685
relprem	56029s	84490s	1.5079
trivdemo	5762s	7304s	1.2676
relinfer	22068s	29839s	1.35214

As far as single articles are concerned, below are tables showing the 10 articles with the longest processing time (in seconds) for each of the utilities. Here is the data collected for `verifier` and `relprem`:

Secs.	verifier	Secs.	New verifier
51	BVFUNC_6	116	GROUP_9
54	GLIB_004	116	AOFA_000
57	JGRAPH_4	117	SCMFSA10
65	SCMISORT	124	JGRAPH_4
85	JORDAN	144	SCMBSORT
90	BINARITH	226	JORDAN
93	BINARI_2	237	SCPQSORT
105	AOFA_I00	240	AOFA_I00
125	SCPQSORT	240	SCMISORT
168	QUATERN2	409	QUATERN2

Secs.	relprem	Secs.	New relprem
913	JORDAN1G	1163	SCPQSORT
973	JORDAN1J	1236	GEOMTRAP
1238	GEOMTRAP	1428	JORDAN
1276	GLIB_005	1434	GLIB_005
1428	SCMFSA10	1537	CONMETR
1485	JORDAN15	1644	AFF_2
1529	CONMETR	1718	JORDAN15
1656	AFF_2	1861	SCMFSA10
1694	JORDAN19	1969	JORDAN19
3972	CONMETR1	3991	CONMETR1

Below is the list of the 10 articles that need the longest time to get processed by `trivdemo` and `relinfer`:

Secs.	trivdemo	Secs.	New trivdemo
85	JORDAN	101	SCPQSORT
108	AOFA_I00	114	JORDAN
113	JORDAN15	117	JORDAN15
118	GEOMTRAP	118	GEOMTRAP
124	JORDAN19	128	JORDAN19
133	GATE_3	133	GATE_3
160	GATE_4	157	GATE_4
161	SCMFSA_2	166	AOFA_I00
200	ANALORT	200	ANALORT
770	XBOOLEAN	763	XBOOLEAN

Secs.	relinfer	Secs.	New relinfer
376	SCPQSORT	425	CONMETR
385	ANALMETR	470	AOFA_I00
442	CONMETR	501	GEOMTRAP
457	JORDAN	567	SCPQSORT
520	GEOMTRAP	663	JORDAN
652	JORDAN19	694	CONMETR1
674	JORDAN15	729	JGRAPH_7
694	CONMETR1	754	JORDAN19
703	JGRAPH_7	770	JORDAN15
919	ANALORT	926	ANALORT

One may notice that in most cases the same articles appear in the top 10 lists for each utility – it shows that the new extension works in a quite predictable way.

Note also that in some cases the running times of the new Checker are even smaller, e.g. `SCMFSA_2` ranking high for `trivdemo` (160 secs.) does not appear in the list of the new `trivdemo`. It is because the new Checker, in this case in `trivdemo`, is able to find much quicker two proofs that can be reduced to a straightforward justification and then it runs for just 40 secs.

The benefit of applying the new Checker was measured by the number of errors reported by the standard utilities after running it on all MML articles and then the number of bytes removed from the library when these errors had been corrected. The experiment was performed on the library "clean" with respect to `relprem` and `trivdemo` (also `inacc` and `chklab`). As there is currently no tool available to handle `relinfer` errors reliably and in a fully automatic way, the results for `relinfer` are calculated as the difference between the number of errors reported by the new and the old version.

Totally, errors were detected in 771 articles: 720 for `relprem`, 261 for `trivdemo` and 637 for `relinfer`. The `relprem` utility detected 8439 errors (there were 296

errors in JGRAPH_4) trivdemo detected 580 + 20 errors[3] (there were 36 errors in TEX_2) and relinfer 6555 (10834 − 4279) errors (also note that there were 392 errors in JGRAPH_4).

Only by automatic elimination of relprem and trivdemo errors, the size of MML was reduced from the original 72826045 bytes to 72587040 bytes. Although the difference (239005 bytes) is only 0.33% of the total size, one may see that with the average size of one MML article equal 71468.2 bytes this makes 3.3 articles. Moreover, with a very rough estimate of 20 bytes reduction per one relinfer error, that would give another 130KB, i.e. two more articles. The most significant changes were done to the articles JGRAPH_4 (4593 bytes removed) and TEX_2 (13483 bytes removed).

All in all, admitting the slowdown in processing time, we claim that the new Checker seems to work in a predictable way and the reduction of Mizar texts obtained thanks to it seems to make up for the loss.

6 Further work

The matching functions of the extended "round-up" algorithm presented in Section 4 have been written to demonstrate how the rounding-up should work rather than provide a final and most efficient solution. Eventually, profiling methods should be used to identify the weak points of the rudimentary implementation and a more efficient code with indexing and other optimizations should be developed to speed up the processing.

The statistics presented in Section 5 show the immediate benefits of running the strengthened Checker on the articles collected in MML at the time of writing this article. But it will also be very important to make a proper use of the new feature when developing new articles as well as during future MML revisions.

It would also be highly desirable to develop tools suitable for processing all MML articles in order to detect and eliminate cases of unnecessary reconsider statements and their proofs, because very often Mizar authors used to state an explicite type restriction only to help the Checker use a specific registration, while with the new "rounding-up" mechanism it is not needed anymore.

And finally, as always when the system has been strengthened, it would be very interesting to turn off the relevant code after some time, and check if authors make use of it in their new developments. Hopefully, extended adjective techniques will become a key feature of the Mizar style of formalizing mathematics.

References

1. Bancerek, G.: On the Structure of Mizar Types. *Electronic Notes in Theoretical Computer Science* **85**(7) (2003).

[3] After running trivdemo there can occur relprem errors (when a whole proof is reduced to a straightforward justification) and new trivdemo errors (when a proof on a higher level is reduced), so that it has to be run several times to get complete results.

2. Kamareddine, F. and R. Nederpelt: A Refinement of de Bruijn's Formal Language of Mathematics. *Journal of Logic, Language and Information* **13**(3) (2004) 287–340.
3. Matuszewski, R. and P. Rudnicki: Mizar: The First 30 Years. *Mechanized Mathematics and Its Applications* **4**(1) (2005) 3–24.
4. Naumowicz, A.: Evaluating Prospective Built-in Elements of Computer Algebra in Mizar. *Studies in Logic, Grammar and Rhetoric* **10**(23) (2007) 191–200.
5. Rudnicki, P. and A. Trybulec: Mathematical Knowledge Management in Mizar. In Buchberger, B. and O. Caprotti (Eds.), *Electronic Proceedings of MKM 2001*. Available at `http://www.emis.de/proceedings/MKM2001/rudnicki.pdf`.
6. Schwarzweller, C.: Mizar Attributes: A Technique to Encode Mathematical Knowledge into Type Systems. *Studies in Logic, Grammar and Rhetoric* **10**(23) (2007) 387–400.
7. Trybulec, A.: Some Features of the Mizar Language. In Proceedings of ESPRIT Workshop, Torino (1993) available on-line at `http://mizar.uwb.edu.pl/project/trybulec93.ps`.
8. Wiedijk, F.: Checker. A compilation of e-mails written by Andrzej Trybulec, available on-line at `http://www.cs.ru.nl/~freek/mizar/by.ps.gz`.

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 18 (31) 2009

The Chinese Remainder Theorem, its Proofs and its Generalizations in Mathematical Repositories

Christoph Schwarzweller

Department of Computer Science, University of Gdańsk
ul. Wita Stwosza 57, 80-952 Gdańsk, Poland
schwarzw@inf.univ.gda.pl

Abstract. In the spirit of mathematical knowledge management theorems are proven with computer assistance to be included into mathematical repositories. In the mathematical literature one often finds not only different proofs for theorems, but also different versions or generalizations with a different background. In mathematical repositories, for obvious reasons, there is usually one version of a theorem with one proof only – the authors choose a version and a proof which can be formalized most easily. In this paper we argue that there are other issues to decide which proof of a theorem or which version of a theorem should be included in a repository. These basically depend on the intended further use of the theorem and the proof. We illustrate these issues in detail with the Chinese Remainder Theorem as an example.

1 Introduction

Over the years much effort has been spent proving more and more elaborated theorems with computer assistance. Some of the most prominent examples recently finished are the Four Colour Theorem using Coq by Georges Gonthier, the Prime Number Theorem using Isabelle/HOL by Jeremy Avigad, and the Jordan Curve Theorem using HOL Light by Tom Hales (shortly after also proven using Mizar by a large group of authors). In contrast, building up repositories of proven theorems is a topic much younger than proving them[1]. Lately this topic has received more and more attention as the area of mathematical knowledge management evolved.

Mathematical knowledge management aims at providing both tools and infrastructure supporting the organization, development, and also teaching of mathematics using modern techniques provided by computers. Consequently, large repositories of mathematical knowledge are here of major interest because they provide users with a data base of – verified – mathematical knowledge. We emphasize the fact that a repository should contain verified knowledge only together with the corresponding proofs. We believe that (machine-checked or -checkable) proofs necessarily belong to each theorem and therefore are an essential part of a repository.

[1] An exception is the Mizar Mathematical Library that goes back to 1989.

ISBN 978-83-7431-229-5 ISSN 0860–150X

However, mathematical repositories should be more than collections of theorems and their proofs accomplished by a prover or proof checker. The overall goal here is not only stating and proving a theorem – though this remains an important and challenging part – but also presenting definitions and theorems so that the "natural" mathematical buildup remains visible. Theories and their interconnections should be available, so that the further development of the repository can be based upon these. Being not trivial as such, this becomes even harder to assure for an open repository with a large number of authors.

In this paper we deal with yet another aspect in building mathematical repositories that is usually not – or only implicitly – taken into account: the evolution of theorems. By this we mean that in mathematics theorems and proofs do not remain stable. New, more elegant proofs for a theorem are found, employing maybe other (new) proof techniques or relying on different new lemmas. Connections between mathematical subfields often lead to a "reformulation" of a theorem in order to apply a different background in the proof. Generalization of theorems also is an issue. The discovery that a theorem holds in a more general case than the original formulation indicates, is a very natural one in mathematics. Hence, both theorems and their proofs very often come in more than one version.

From the mathematical point of view this is not only harmless but also desirable; it is part of the mathematical progress. In mathematical repositories, however, one usually finds only one version of a theorem with one proof only. This is of course reasonable, if formalizing – that is stating and proving the theorem in a system – is the major goal. At this point sometimes different versions of theorems or proofs are implicitly dealt with. Before starting the formalization alternatives are checked and the one (seemingly) being best suited for formalization is chosen.

The argument "Once we have the theorem included in a repository, we can use it for further development, no matter how it has been proven" however, excludes technical problems of building mathematical repositories. When extending the repository it might occur that to finish a proof relying on a special theorem, it would be much better to use another version of this theorem. And if repositories are used for teaching mathematics, different versions of theorems and proofs are of course of interest for didactic reasons.

Consequently, it may be reasonable to include different versions of theorems or proofs in mathematical repositories, though at first sight this seems not only to blow up the repository, but also causes additional work. Such decisions, therefore, will usually depend on special intentions of the theorem, e.g.: Will it be also used for didactic purposes? Can other versions be formalized with a reasonable amount of work? In the following we will illustrate the above considerations by using the Chinese Remainder Theorem [16,8] as a concrete example.

The plan of the paper is as follows. In the next section we present the Chinese Remainder Theorem (CRT) in its standard version together with three different proofs. To give the reader a feeling about formalizing such a theorem, we give a brief introduction to the Mizar system [21] in Section 3 and a glimpse on the Mizar proof of the CRT in Section 4. Section 5 then deals with other versions of the CRT. One version is rather technical – due to formalization issues in open mathematical repositories. The other one is a completely different formulation of the theorem

which, unlike the first one, relies more on abstract algebra. Section 6 finally is devoted to possible generalizations of the CRT and their relation to the original theorem.

2 The Chinese Remainder Theorem and Its Proofs

The CRT is a result about congruences over the integers. It states that an integer u can be completely described by the sequence of its remainders – if the number of remainders is big enough. The "standard" version of the theorem reads as follows.

Theorem 1. Let $m_1, m_2, \ldots, m_r$ be positive integers such that m_i and m_j are relatively prime for $i \neq j$. Let $m = m_1 m_2 \cdots m_r$ and let $u_1, u_2, \ldots, u_r$ be integers. Then there exists exactly one integer u with

$$0 \leq u < m \quad \text{and} \quad u \equiv u_i \bmod m_i \quad \text{for all} \quad 1 \leq i \leq r. \quad \diamond$$

That this theorem should be part of a mathematical repository needs no further explanation. But what kinds of proofs exist and what reasons are there to include these proofs in mathematical repositories? In the following we present three different proofs of the theorem and discuss their relevance to be included in mathematical repositories. It is very easy to show, that there exists at most one such integer u; in the following proofs we therefore focus on proving the existence of u. The proofs are taken from [16].

First proof: Suppose integer u runs through the m values $0 \leq u < m$. Then $(u \bmod m_1, \ldots, u \bmod m_r)$ also runs through m different values, because the system of congruences has at most one solution. Because there are exactly $m_1 m_2 \cdots m_r = m$ different tuples $(v_1, \ldots, v_r)$ with $0 \leq v_i < m_i$, every tuple occurs exactly once, and hence for one of those we have $(u \bmod m_1, \ldots, u \bmod m_r) = (u_1, \ldots, u_r)$. $\diamond$

This proof is pretty elegant and uses a rather obvious variant of the pigeon hole principle: If we pack m items without repetition to m buckets, then we must have exactly one item in each bucket. It is therefore valuable to include this proof in a repository for didactic or aesthetic reasons. On the other hand, formalization of the proof is not straightforward. This proof is one of those, that can technically really blow up being formalized. One has to argue about the number of different r-tuples and, more importantly, to show that there exists a bijection between the set of r-tuples and the non-negative integers smaller than m. Another disadvantage is that the proof is non-constructive, so that it gives no hints to find the value of u – besides the rather valueless "Try and check all possibilities, one will fit". This is even more disturbing, because a constructive proof can easily be given:

Second proof: We can find integers M_i for $1 \leq i \leq r$ with

$$M_i \equiv 1 \bmod m_i \quad \text{and} \quad M_j \equiv 0 \bmod m_i \quad \text{for } j \neq i.$$

Because m_i and m/m_i are relatively prime, we can take for example

$$M_i = (m/m_i)^{\varphi(m_i)},$$

where φ denotes the Euler function. Now,

$$u = (u_1 M_1 + u_2 M_2 + \cdots + u_r M_r) \bmod m$$

has the desired properties. $\diamond$

This proof constructs r constants M_i with which the sought-after u can easily be computed. It therefore, in some sense, contains more information than the first proof, that should be contained in the repository also. The proof uses far more evolved mathematical notations – namely Euler's function – and for that reason may also be considered more interesting than the first one. Formalization requires the use of Euler's function[2] which may lead to a lot of preliminary work. From a computer science point of view the proof has two disadvantages. First, it is not easy to compute Euler's function; in general one has to decompose the moduli m_i into their prime factors. Second, the M_i being multiples of m/m_i are really big numbers, so that a better method for computing u is highly desirable. Such a method has indeed been found by H. Garner, which gives a third proof of Theorem 1:

Third proof: Because we have $\gcd(m_i, m_j) = 1$ for $i \neq j$ we can find integers c_{ij} for $1 \leq i < j \leq r$ with

$$c_{ij} m_i \equiv 1 \bmod m_j$$

by applying the extended Euclidean algorithm to m_i and m_j. Now taking

$$\begin{aligned}
v_1 &:= u_1 \bmod m_1 \\
v_2 &:= (u_2 - v_1)c_{12} \bmod m_2 \\
v_3 &:= ((u_3 - v_1)c_{13} - v_2)c_{23} \bmod m_3 \\
&\vdots \\
v_r &:= (\ldots((u_r - v_1)c_{1r} - v_2)c_{2r} - \cdots - v_{r-1})c_{(r-1)r} \bmod m_r
\end{aligned}$$

and then setting

$$u := v_r m_{r-1} \cdots m_2 m_1 + \cdots + v_3 m_2 m_1 + v_2 m_1 + v_1$$

we get the desired integer u. $\diamond$

The proof uses $\binom{r}{2}$ constants c_{ij} that can be computed with the extended Euclidean algorithm because we have $\gcd(m_i, m_j) = 1$ for $i \neq j$. When constructing the v_i the application of the modulo operation in each step ensures that the occurring values remain small. Note that the finally computed u actually is a radix number in $m_1, m_2, \ldots, m_r$. The proof is far more technical than the others in constructing $\binom{r}{2} + r$ additional constants, the v_i in addition being recursively defined. Therefore, a formalization of this proofs will be rather unpleasant. On the other hand, however, this proof includes an efficient method to compute the integer u from Theorem 1.

[2] Actually this is not completely true: a mild modification of the proof does without Euler's function; compare Section 4.2.

We see that the question which proof of a theorem should be formalized, does not only depend on the hardness of the formalization in a given system. Both elegance and the amount of information are issues that can be taken into consideration – this may even result in formalizing more than one proof. This, however, is not the end of the line. In the literature we often find different versions or even generalizations of a theorem, and again the question is which one to choose. Before we discuss these issues in case of the CRT, we will briefly provide insight into how to formalize the theorem in the Mizar system [21].

3 The Mizar System

The Mizar language as well as the Mizar system have been described elsewhere (see e.g. [24,23,25,32]). Here we only give a brief overview necessary to follow the subsequent sections.

Mizar's [24,21] logical basis is the classical first order logic extended with so-called schemes. Schemes allow for free second order variables, in this way enabling, for example, the definition of induction schemes. The current development of the Mizar Mathematical Library (MML) is based on Tarski-Grothendieck set theory (a variant of Zermelo-Fraenkel set theory using Tarski's axiom on arbitrarily large, strongly inaccessible cardinals [28] which can be used to prove the axiom of choice), though in principle the Mizar language allows for other axiom systems also. Mizar proofs are written in the natural deduction style similar to the calculus of [13]. The rules of the calculus are connected with corresponding (English) natural language phrases, so that the Mizar language is close to the one used in mathematical textbooks. The Mizar proof checker verifies the individual proof steps using the notion of obvious inferences [5] to shorten the rather long proofs of pure natural deduction.

Mizar objects are typed, the types forming a hierarchy with the fundamental type `set` [1]. New types are constructed using type constructors called modes. Modes can be decorated with adjectives – given by so-called attribute definitions – in this way extending the type hierarchy. For example, given the mode `Ring` and an attribute `commutative` a new mode `commutative Ring` can be constructed, which obeys all the properties given by the mode `Ring` plus the ones stated by the attribute `commutative`. Furthermore, a variable of type `commutative Ring` is also of type `Ring`, which implies that all notions defined for `Ring` are available for `commutative Ring`. In addition all theorems proved for type `Ring` are applicable for objects of type `commutative Ring`; indeed the Mizar checker itself infers subtype relations in order to check whether theorems are applicable for a given type.

4 A Mizar Formalization of the Chinese Remainder Theorem

We believe that in mathematical repositories both the formalization of the theorem itself – that is its representation and in particular its readability – and the formalization of the proof are equally important. Therefore, first we discuss our

representation of the theorem and then describe how we formalized the second proof of Section 2.

4.1 The Theorem

We represent the given integers $u_1, \ldots, u_r$ and $m_1, \ldots, m_r$ as finite sequences over the integers. For the m_i we need the additional condition that they are pairwise relatively prime. Using the already defined predicate `are_relative_prime` [18] this property is introduced in a Mizar attribute definition:

```
definition
  let f be integer-yielding FinSequence;
  attr f is Chinese_Remainder means
  for i,j being natural number
  st i in dom f & j in dom f & i <> j holds f.i, f.j are_relative_prime;
end;
```

Then a finite sequence of integers m_i fulfilling the assumption of Theorem 1 – a Chinese remainder sequence – can be described by the following mode definition. Note that the number r of moduli is given simply by the length of the sequence.

```
definition
  mode CR_Sequence is non empty positive-yielding Chinese_Remainder
          (integer-yielding FinSequence);
end;
```

As a consequence each element of type `CR_sequence` describes a set of moduli m_i fulfilling the assumptions of Theorem 1. Note also that due to Mizar's type widening mechanism all theorems proven for elements of type `FinSequence` and `integer-yielding FinSequence` can also be automatically applied to elements of type `CR_Sequence`.

Using the predicate `are_congruent_mod` [30] and the functor `Product` [2] giving the product of the elements of a finite sequence it is now easy to state the CRT in Mizar in a very readable fashion. Here we present only the theorem on the existence of the integer u. A second theorem describing the uniqueness of u is straightforward.

```
theorem
  for u being integer-yielding FinSequence,
      m being CR_Sequence st len u = len m
  ex z being Integer
    st 0 <= z & z < Product(m) & for i being natural number
    st i in dom u holds z,u.i are_congruent_mod m.i;
```

4.2 A Mizar Formalization of the Second Proof

The proof starts with the definition of the constants M_i. Instead of Euler's function we use a variant that analogously to Garner's proof finds the constants by employing the extended Euclidean algorithm: If m is the product of the m_i we

have $\gcd(m/m_i, m_i) = 1$ for $i = 1, \ldots, r$. We can therefore find integers s_i with $s_i * (m/m_i) \equiv 1 \bmod m_i$ and thus constants M_i with the desired properties. In Mizar we defined a finite sequence holding the values of M_i for $i = 1, \ldots, r$:

```
definition
  let m be CR_Sequence;
  mode CR_coefficients of m -> FinSequence means
    len it = len m &
    for i being natural number st i in dom it holds
      ex s being Integer, mm being Integer
      st mm = Product(m)/m.i & s*mm,1 are_congruent_mod m.i &
         it.i = s * (Product(m)/m.i);
end;

registration
  let m be CR_Sequence;
  cluster -> integer-valued CR_coefficients of m;
end;
```

To be more precise, we actually do not fix the constants s_i, we just state (and prove) their existence. Therefore every finite sequence using appropriate values for the s_i is of type `CR_coefficients`, and not only the one with the value for the s_i computed by the extended Euclidean algorithm. As a consequence all that follows holds for arbitrary choices of the s_i.

The next step is to construct the integer $u = (u_1 M_1 + \ldots + u_r M_r) \bmod m$ from the constants M_i. This again is straightforward using the Mizar functors `(#)` for componentwise multiplication of sequences [22] and `Sum` for summing up the elements of a sequence [2]. We defined the Mizar functor `to_int` as follows.

```
definition
  let u be integer-yielding FinSequence,
      m be CR_Sequence such that len m = len u;
  func to_int(u,m) -> Integer means :Def_to_int:
    for c being CR_coefficients of m holds it = Sum(u(#)c) mod Product(m);
end;
```

Note that the term `Sum(u(#)c) mod Product(m)` denotes the same integer for an arbitrary choice of the coefficients in `c`; or in other words $(u_1 * M_1 + \ldots + u_r M_r) \bmod m$ always gives the same integer u no matter what concrete values s_i have been used in the construction of the `CR_sequence c`. This in fact is the reason that we can state `to_int(u,m)` in this way as a Mizar functor.

The proof of the CRT now consists of showing that `to_int(u,m)` has the desired properties. The key property here is of course that the constructed sum `Sum(u(#)c)` is congruent with u_i modulo m_i for all $i = 1, \ldots, r$ as stated in the following theorem.

```
theorem congsum:
  for u being integer-valued FinSequence,
      m being CR_Sequence st len m = len u
```

```
  for c being CR_coefficients of m,
      i being natural number st i in dom m
    holds Sum(u(#)c),u.i are_congruent_mod m.i;
```

The proof is pretty technical and shows by induction on the length l of the sequence `u(#)s` that its sum is congruent to 0 if $l < i$ and congruent to m_i if $l \geq i$. Together with the following result from [30]

```
theorem :: INT_1:41
  for i1,i2,i3,i4,i5 being Integer
  holds i4 * i5 = i3
  implies (i1,i2 are_congruent_mod i3 implies i1,i2 are_congruent_mod i4);
```

the rest of the proof basically consists of applying properties of integer congruences, that is applying theorems already present in the Mizar Mathematical Library. The obvious fact that $0 \leq$ `to_int(u,m)` $<$ `Product(m)` has been shown in `lemma2`. To give the reader an impression of how Mizar proofs are written, we include the proof:

```
proof
let u be integer-yielding FinSequence, m be CR_Sequence;
assume AS: len u = len m;
take z = to_int(u,m);
now let i be natural number;
  assume A: i in dom u;
  consider c being CR_coefficients of m;
  set s = Sum(u(#)c) mod Product(m);
  B: dom m = Seg(len u) by AS,FINSEQ_1:def 3
          .= dom u by FINSEQ_1:def 3;
  then Sum(u(#)c),u.i are_congruent_mod m.i by A,AS,congsum;
  then C: Sum(u(#)c) mod m.i = u.i mod m.i by INT_3:12;
  dom m = Seg(len u) by AS,FINSEQ_1:def 3
       .= dom u by FINSEQ_1:def 3;
  then m.i in rng m by A,FUNCT_1:12;
  then D: m.i > 0 by PARTFUN3:def 1;
  consider y being Integer such that E: y * m.i = Product(m) by A,B,thm;
  s mod Product(m) = Sum(u(#)c) mod Product(m) by INT_3:13;
  then s,Sum(u(#)c) are_congruent_mod Product(m) by INT_3:12;
  then s,Sum(u(#)c) are_congruent_mod m.i by E,INT_1:41;
  then s mod m.i = Sum(u(#)c) mod m.i by INT_3:12;
  then s,u.i are_congruent_mod m.i by D,C,INT_3:12;
  hence z,u.i are_congruent_mod m.i by Def_to_int,AS;
  end;
hence thesis by AS,lemma2;
end;
```

We would like to add that using the functor `to_int` one can easily show that modular integer arithmetic based on the CRT is correct [26]. Modular integer arithmetic performs integer arithmetic by transforming integers into sequences of their remainders, that is an integer u is transformed into an r-tuple of the form $(u \bmod m_1, \ldots, u \bmod m_r)$ for given moduli $m_1, m_2, \ldots, m_r$. After performing the

arithmetic operation componentwise on the tuples the CRT (via `to_int`) allows to transform the result back to the ordinary integers without loss of information – if $m = m_1 m_2 \cdots m_r$ is big enough.

To prove this we can easily define a functor `mod` that transforms an integer `u` into the sequence of `u`'s remainders. The moduli m_i are here given by a finite sequence `m`.

```
definition
  let u be Integer,
      m be integer-yielding FinSequence;
  func mod(u,m) -> FinSequence means
    len it = len m &
    for i being natural number st i in dom it holds it.i = u mod m.i;
end;
```

The componentwise arithmetic operations are then just the operations for sequences given in [22]. So, for example, `mod(u,m) (#) mod(v,m)` describes the componentwise multiplication of remainders for the integers `u` and `v`. Translating this finite sequence to an integer `z` using the functor `to_int` from above results in `z = u * v`, if `u * v < Product(m)`. We thus get the following

```
theorem
  for u,v being Integer,
      m being CR_Sequence st 0 <= u * v & u * v < Product(m)
    holds to_int(mod(u,m) (#) mod(v,m), m) = u * v;
```

Analogous theorems for addition and subtraction of integers can be easily stated and proven (see [26]).

5 Other Versions of the Chinese Remainder Theorem

In this section we consider other versions of the CRT, that is, generally speaking, other theorems stating the same fact as the original theorem. We try to analyze where these different versions come from and, more importantly, their impact on mathematical repositories.

5.1 Another Mathematical Version

The most natural reason for different versions of theorems is of course that mathematicians often look at the same issue from different perspectives. The CRT presented in Section 2 deals with congruences over the integers; it states the existence of an integer solving a given system of congruences. Looking from a more algebraic point of view, that is concentrating on the structures being involved aside from the integers, we see that the moduli m_i can be interpreted as describing the residue class rings $\mathcal{Z}_{m_i}$. The existence and uniqueness of the integer u from the CRT then gives rise to an isomorphism between these rings. So the reformulated CRT looks as follows [8].

Theorem 2. Let $m_1, m_2, \ldots, m_r$ be positive integers such that m_i and m_j are relatively prime for $i \neq j$ and let $m = m_1\, m_2 \cdots m_r$. Then we have the ring isomorphism

$$\mathcal{Z}_m \;\cong\; \mathcal{Z}_{m_0} \times \cdots \times \mathcal{Z}_{m_r}. \;\diamond$$

It is not easy to decide which version of the CRT is better suited for inclusion in a mathematical repository. Theorem 2 looks more elegant and in some sense contains more information than Theorem 1: It does not state the existence of a special integer, but the equality of two mathematical structures.[3] The proof of Theorem 2 uses the homomorphism theorem for rings and is therefore interesting for didactic reasons, too.

On the other hand, Theorem 1 uses integers and congruences only, so that one needs less preliminaries to understand it. Theorem 1 and its proof also give more information than Theorem 2 concerning computational issues[4] – at least if not the first proof only has been formalized.

5.2 Another Technical Version

Another reason for additional versions of a theorem may be based in the mathematical repository itself. Here again especially open repositories play an important role. Different styles of formalizing and different kinds of mathematical understanding and preferences meet in one repository. So, it may happen that two authors formalize the same (mathematical) theorem, but choose a different formulation and/or a different proof. We call this technical or representational versions.

In the Mizar Mathematical Library in [17], for example, we find another version of the CRT from Section 2:

```
theorem :: WSIERP_1:44
  len fp>=2 &
  (for b,c st b in dom fp & c in dom fp & b<>c holds (fp.b gcd fp.c)=1)
    implies
  for fr st len fr=len fp holds
  ex fr1 st (len fr1=len fp &
  for b st b in dom fp holds (fp.b)*(fr1.b)+(fr.b)=(fp.1)*(fr1.1)+(fr.1));
```

This theorem is not immediately understandable, also because the variables' types are not explicitly stated but given by a so-called reservation – that in the article of course occurs before the theorem.

```
reserve b,c for Nat,
        fp for FinSequence of NAT,
        fr,fr1 for FinSequence of INT;
```

[3] Of course this equality easily follows from Theorem 1, but is not explicitly stated there.

[4] To apply the homomorphism theorem in the proof of Theorem 2 one needs to show that the canonical homomorphism is a surjection with kernel (m). This sometimes is done by employing the extended Euclidean algorithm, so that this proof gives an algorithm, too.

In this version no attributes are used. The condition that the m_i are pairwise relatively prime is here stated explicitly using the `gcd` functor for natural numbers. Also the congruences are described arithmetically: $u \equiv u_i \bmod m_i$ means that there exists a x_i such that $u = u_i + x_i * m_i$, so the theorem basically states the existence of $x_1, \ldots, x_r$ instead of u.

Since the article has been written more than 10 years ago, a reason for this technical formulation is hard to find. It may be that at the time of writing Mizar's attribute mechanism was not so far developed as today, i.e. the author reformulated the theorem in order to get it formalized at all. Another explanation for this second technical version might be that the author when formalizing the CRT already had in mind a particular application and therefore chose a formulation better suited to prove the application.

We see that in general the way authors use open systems to formalize theorems has a crucial impact on the formulation, that is on the technical version of a theorem – and may lead to different versions of the same theorem.

6 Generalizations of the Chinese Remainder Theorem

Generalization of theorems is everyday occurrence in mathematics. In the case of mathematical repositories generalization is a rather involved topic: It is not obvious whether the less general theorem can be eliminated. Proofs of other theorems using the original version might not work automatically with the more general theorem instead. The reason may be that a slightly different formulation or even a different (mathematical or technical) version of the original theorem has been formalized. Then the question is: Should one rework all these proofs or keep both the original and the more general theorem in the repository? To illustrate that this decision is both not trivial and important for the organization of mathematical repositories we present in this section some generalizations of the CRT.

6.1 Mathematical Generalizations

A rather mild generalization of Theorem 1 is based on the observation that the range in which the integer u lies, does not need to be fixed. It is sufficient that it has the width $m = m_1 m_2 \cdots m_r$. This easily follows from the properties of the congruence $\equiv$.

Theorem 3. Let $m_1, m_2, \ldots, m_r$ be positive integers such that m_i and m_j are relatively prime for $i \neq j$. Let $m = m_1\, m_2 \cdots m_r$ and let $a, u_1, u_2, \ldots, u_r$ be integers. Then there exists exactly one integer u with

$$a \leq u < a + m \text{ and } u \equiv u_i \bmod m_i$$

for all $1 \leq i \leq r$. $\diamond$

It is trivial that for $a = 0$ we get the original Theorem 1. The old proofs can very easily be adapted to work with this generalization of the theorem. Maybe

the system checking the repository even automatically infers that Theorem 3 with $a = 0$ substitutes the original theorem. If not, however, even the easy changing all the proofs to work with the generalization can be an extensive, unpleasant, and time-consuming task.

A second generalization of the CRT is concerned with the underlying algebraic structure. The integers are the prototype example for Euclidean domains. Taking into account that the residue class ring $\mathcal{Z}_n$ in fact is the factor ring of $\mathcal{Z}$ by the ideal $n\mathcal{Z}$, it is rather obvious that the following generalization[5] holds.

Theorem 4. Let R be a Euclidean domain. Let $m_1, m_2, \ldots, m_r$ be positive integers such that m_i and m_j are relatively prime for $i \neq j$ and let $m = m_1\, m_2 \cdots m_r$. Then we have the ring isomorphism

$$R/(m) \;\cong\; R/(m_0) \times \cdots \times R/(m_r). \;\diamond$$

This generalization may cause some problems: In mathematical repositories it is an immense difference whether one argues about the set of integers (with the usual operations) or the ring of integers: They have just different types.[6] Technically, this means that in mathematical repositories we often have two different representations of the integers. In the mathematical setting theorems of course hold for both of them. However, proofs using one representation will not work for the other one. Consequently, though Theorem 4 is more general, it will not work for proofs using integers instead of the ring of integers; for that a similar generalization of Theorem 1 is necessary. So in this case in order to make all proofs work with a generalization, we need to provide generalizations of different versions of the original theorem – or just change the proofs with the "right" representation leading to an unbalanced organization of the repository.

We close this subsection with a generalization of the CRT that abstracts even from algebraic structures. The following theorem [19] deals with sets and equivalence relations only and presents a condition whether the "canonical" function σ is onto.

Theorem 5. Let α and β be equivalence relations on a given set M. Let $\sigma : M \longrightarrow M/\alpha \times M/\beta$ be defined by $\sigma(x) := (\alpha(x), \beta(x))$. Then we have $\ker(\sigma) = \alpha \cap \beta$ and σ is onto if and only if $\alpha \circ \beta = M \times M$. $\diamond$

In this generalization almost all of the familiar CRTs gets lost. There are no congruences, no algebraic operations, only the factoring (of sets) remains. Therefore, it seems hardly possible to adapt proofs using any of the preceding CRTs to work with this generalization in a reasonable amount of time. Any application will rely on much more concrete structures, so that too much effort has to be spent to adapt a proof. Theorem 5 in some sense is too general to reasonably work with. However, even if not applicable, the theorem stays interesting from a didactic point of view.[7] It illustrates how far we sometimes can generalize and may provide the

[5] Literally this is a generalization of Theorem 2, but of course Theorem 1 can be analogously generalized to Euclidean domains.

[6] Though often the ring of integers is constructed using the (set of) integers.

[7] In fact the proof of Theorem 5 has been an exercise in lectures on linear algebra.

starting point of a discussion whether this is – aside from mathematical aesthetics – expedient; a topic that is also of great interest for the organization of mathematical repositories.

6.2 Generalization towards Concepts

Another kind of possible generalization does not stem from mathematics, but from the intention of bringing more structure into the organization of mathematical repositories and can be compared with the ideas of [27,7] on theory interpretation. Modular arithmetic from Section 4.2 using the CRT is a typical example for the concept of computing with homomorphic images: transforming data into another algebraic structure, performing there algebraic operations and then reconstructing the result in the original structure; or, to say it a little bit shorter, performing the algebraic operations in another representation.

A second example is the multiplication of polynomials using the discrete Fourier transformation [8]. A discrete Fourier transformation DFT_ω changes the representation of a polynomial p with $\deg(p) < n$ to an n-tuple by evaluating p for n values. A special choice of these n values – $\omega^0, \omega^1, \ldots, \omega^{n-1}$ for an n-th primitive root ω – ensures that no information is lost. After carrying out a componentwise multiplication on the n-tuples another discrete Fourier transformation – in fact $(\mathrm{DFT}_\omega)^{-1}$ – constructs the result polynomial.

Though different in nature, both the CRT and the discrete Fourier transformation serve to prove that a special instantiation of the concept computing with homomorphic images is correct. Putting it the other way round, both of them can in some sense be generalized to the concept computing with homomorphic images. This adjacency should be reflected in mathematical repositories. This gives rise to a theorem describing in essence the correctness of the concept.[8]

Theorem 6. Let R an S be <structures>, and let $f : R \longrightarrow S$ be a function. Let u and v be elements of r. If f, u and v fulfill <a condition>, then we have $u\ op_R\ v = f^{-1}(f(u)\ op_S\ f(v))$ for all operations op_R in R. $\diamond$

Note that the usual homomorphic condition $f(u\ op_R\ v) = f(u)\ op_S\ f(v)$ does not allow to transform the result back into the structure R. If f is an isomorphism, then of course the theorem becomes trivial. It therefore would be interesting to find weaker realizations of <a condition>.

Theorem 6 states (in an abstract way) that to perform operations in R the change of the representation from R to S using f is admissible if <a condition> holds. Having such a theorem in a mathematical repository would allow to handle the concept of computing with homomorphic images by just showing that a special case fulfills <a condition>. Modular integer arithmetic, for example, according to Theorem 2 sets R to $\mathcal{Z}_m$ and S to $\mathcal{Z}_{m_0} \times \cdots \times \mathcal{Z}_{m_r}$. It then takes the `mod` functor as f and the `to_int` functor as f^{-1}. Another variant according to Theorem 1 is setting R to $\mathcal{Z}$ instead of $\mathcal{Z}_m$. In this case however we get the additional condition

[8] The following "theorem" holds for arbitrary (algebraic) structures and a yet to find condition, which we indicated by using Backus-Naur-like brackets < and >.

that $u\ op_R\ v < m$ holds. In the case of multiplication of polynomials p and q we get that $f = \mathrm{DFT}$ and $f^{-1} = \mathrm{DFT}^{-1}$ with the additional condition that $\deg(p*q) < n$.

7 Related Work

Formalizations of the CRT can be found in other systems besides Mizar. The first computer proof of the CRT dates back to 1992 using Rewrite Rule Laboratory [15]. Because the rewriting approach cannot handle quantifiers (all variables are assumed to be universally quantified), existential quantifiers are eliminated by introducing Skolem functions. So in [33] we find the following CRT.

$$\begin{array}{l}(allpositive(Y) \wedge allprime(Y)) \Rightarrow \\ \quad (allcongruent(soln(Y), Y) \wedge \\ \quad\quad ((allcongruent(x_1, Y) \wedge allcongruent(x_2, Y)) \Rightarrow \\ \quad\quad\quad\quad (rem(x_1 - x_2, products(Y)) = 0)))\end{array}$$

where *soln* is the above mentioned Skolem function. The goal was to experiment with the cover set induction principle implemented in Rewrite Rule Laboratory, that is the challenging point was proving the theorem – in whatever formulation.

In more current proof systems the CRT also have been formalized. Interestingly, we found only two-number versions, that is CRTs where the number of moduli is restricted to two. In HOL Light [11], for example, we find such a version of Theorem 1 stating that in case of two moduli `a` and `b` there exists a simultaneous solution `x` of the congruences.

```
# INTEGER_RULE
  '!a b u v:int. coprime(a,b) ==>
                       ?x. (x == u) (mod a) /\ (x == v) (mod b)';;
```

`INTEGER_RULE` is a rule for proving divisibility properties of the integers. The rule is partly heuristic and most of the statements automatically proven with it are universally quantified. Again the main purpose of the CRT is to illustrate the power of a proof technique.

The CRT has been formalized in `hol98`, too [12]. Here we find a two-number version of Theorem 2 that in addition is restricted to multiplicative groups. Technically, the theorem states that for moduli p and q the function $\lambda x.(x \bmod p, x \bmod q)$ is a group isomorphism between $\mathcal{Z}_{pq}$ and $\mathcal{Z}_p \times \mathcal{Z}_q$.

$$\begin{array}{l}\vdash \forall p, q. \\ \quad 1 < p \wedge 1 < q \wedge \mathbf{gcd}\ p\ q = 1 \Rightarrow \\ \quad (\lambda x.(x \bmod p, x \bmod q)) \in \\ \quad\quad \mathbf{group\_iso}\ (\mathbf{mult\_group}\ pq) \\ \quad\quad (\mathbf{prod\_group}\ (\mathbf{mult\_group}\ p)\ (\mathbf{mult\_group}\ q))\end{array}$$

Note that, in contrast to Theorem 2, the isomorphism is part of the theorem itself and not hidden in the proof. The main goal of [12] was the verification of the Miller-Rabin probabilistic primality test. Therefore the restriction to multiplicative groups is reasonable, because this version of the CRT is sufficient for the verification.

In the Coq Proof Assistant [3] the CRT has been proved for a bit vector representation of the integers [20]. We see that this again is a version of Theorem 1 restricted to two moduli `a` and `b`.

```
Theorem chinese_remaindering_theorem :
 forall a b x y : Z,
 gcdZ a b = 1%Z -> {z : Z | congruentZ z x a /\ congruentZ z y b}.
```

In fact this theorem and its proof are the result of rewriting a former proof of the CRT in Coq. So in Coq there exist two versions of the CRT – though the former one has been declared obsolete.

8 Conclusions

In order to discuss the question which proof and which version of a theorem is best suited for inclusion in mathematical repositories, we have presented various proofs, versions and generalizations of the Chinese Remainder Theorem.

It is probably no surprise that each version or generalization comes with its pros and cons, so that it seems not possible in general to decide which one is best suited to be included in a repository. On the contrary it may even be reasonable to include more than one proof or version convenient for different purposes. One maybe is better suited for further development (of applications) of the repository, whereas another one better for didactic reasons.

It is not foreseeable whether it is possible to develop criteria for deciding which proof, version or generalization of a theorem to include in a mathematical repository. However, it might be that the attempt to do so – like hopefully the considerations in this paper – is a step towards the development of schemata how to organize mathematical repositories.

Acknowledgment: I would like to thank the reviewers for their detailed comments which have greatly improved the presentation of the paper.

References

1. Bancerek, G.: *On the Structure of Mizar Types*; in: H. Geuvers and F. Kamareddine (eds.), Proceedings of MLC 2003, ENTCS 85(7), 2003.
2. Byliński, C.: *The Sum and Product of Finite Sequence of Real Numbers*; in: Formalized Mathematics, 1(4), pp. 661–668, 1990.
3. *The Coq Proof Assistant*; available at `http://coq.inria.fr`, 2008.
4. Davenport, J.H.: *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, pp. 17–29, 2003.
5. Davies, M.: *Obvious Logical Inferences*; in: Proceedings of the 7th International Joint Conference on Artificial Intelligence, pp. 530–531, 1981.
6. de Bruijn, N.G.: *The Mathematical Vernacular, a language for mathematics with typed sets*; in: P. Dybjer et al. (eds.), Proceedings of the Workshop on Programming Languages, Marstrand, Sweden, 1987.

7. Farmer, W., Guttman, J. and Thayer, F.: *IMPS: An Interactive Mathematical Proof System*; in: Journal of Automated Reasoning 11, 213–248, 1993.
8. von zur Gathen, J. and Gerhard, J.: *Modern Computer Algebra*; Cambridge University Press, 1999.
9. Grabowski, A. and Schwarzweller, C.: *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, pp. 49–64, 2006.
10. Graham, R.E., Knuth, D.E. and Patashnik, O.: *Concrete Mathematics*; Addison-Wesley, 1994.
11. Harrison, J.: *The HOL Light System Reference*; available at `http://www.cl.cam.ac.uk/~jrh13/hol-light/reference_220.pdf`, 2008.
12. Hurd, J.: *Verification of the Miller-Rabin Probabilistic Primality Test*; in: Journal of Logic and Algebraic Programming, 50(1-2), pp. 3–21, 2003.
13. Jaśkowski, S.: *On the Rules of Supposition in Formal Logic*; in: Studia Logica, vol. 1, 1934.
14. Kamareddine, F. and Nederpelt, R.: *A Refinement of de Bruijn's Formal Language of Mathematics*; in: Journal of Logic, Language and Information, 13(3), pp. 287–340, 2004.
15. Kapur, D. and Zhang, H.: *An Overview of Rewrite Rule Laboratory (RRL)*; in: N. Dershowitz (ed.), Proceedings of the 3rd International Conference on Rewriting Techniques and Applications, LEcture Notes in Computer Science 355, pp. 559–563, 1989.
16. Knuth, D.: *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*; 3rd edition, Addison-Wesley, 1997.
17. Kondracki, A.: *The Chinese Remainder Theorem*; in: Formalized Mathematics, 6(4), pp. 573–577, 1997.
18. Kwiatek, R. and Zwara, G.: *The Divisibility of Integers and Integer Relative Primes*; in: Formalized Mathematics, 1(5), pp. 829–832, 1990.
19. Lüneburg, H.: *Vorlesungen über Lineare Algebra* (in German), BI Wissenschaftsverlag, 1993.
20. Ménissier-Morain, V.: *A Proof of the Chinese Remainder Lemma*; available at `http://logical.saclay.inria.fr/coq/distrib/current/contribs/ZChinese.html`, 2008.
21. The Mizar Home Page, `http://mizar.org`, 2009.
22. The Mizar Mathematical Library Committee, *Properties of Number-Valued Functions*, 2007.
23. Naumowicz, A. and Byliński, C.: *Improving Mizar Texts with Properties and Requirements*, in: A. Asperti, G. Bancerek, and A. Trybulec (eds.), Proceedings of the 3rd International Conference on Mathematical Knowledge Management, Lecture Notes in Computer Science 3119, pp. 190–301, 2004.
24. Rudnicki, P. and Trybulec, A.: *Mathematical Knowledge Management in Mizar*; in: B. Buchberger, O. Caprotti (eds.), Proceedings of the 1st International Conference on Mathematical Knowledge Management, Linz, Austria, 2001.
25. Schwarzweller, C.: *Mizar Attributes: A Technique to Encode Mathematical Knowledge into Type Systems*; in: Studies in Logic, Grammar and Rhetoric, vol. 10(23), pp. 387–400, 2007.
26. Schwarzweller, C.: *Modular Integer Arithmetic*; in: Formalized Mathematics, 16(3), pp. 247–252, 2008.
27. Shoenfield, J.: *Mathematical Logic*; Addison-Wesley, 1967.

28. Tarski, A.: *On Well-Ordered Subsets of Any Set*; in: Fundamenta Mathematicae, vol. 32, pp. 176–183, 1939.
29. Treyderowski, K. and Schwarzweller, C.: *Multiplication of Polynomials using Discrete Fourier Transformation*; in: Formalized Mathematics, 14(4), pp. 121–128, 2006.
30. Trybulec, M.: *Integers*; in: Formalized Mathematics, 1(3), pp. 501–505, 1990.
31. Wiedijk, F.: *On the Usefulness of Formal Methods*; Nieuwsbrief van de NVTI, pp. 14–23, 2006.
32. Wiedijk, F.: *Writing a Mizar Article in Nine Easy Steps*; available at `http://www.mizar.org/project/bibliography.html`, 2008.
33. Zhang, H. and Hua, X.: *Proving the Chinese Remainder Theorem by the Cover Set Induction*; in: D. Kapur (ed.), Automated Deduction – CADE-11, Proceedings of the 11th International Conference on Automated Deduction, Lecture Notes in Computer Science 607, pp. 431–455, 1992.

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 18 (31) 2009

Combining Mizar and TPTP Semantic Presentation and Verification Tools

Josef Urban[1]*, Geoff Sutcliffe[2], Steven Trac[2], and Yury Puzis[2]

[1] Charles University, Czech Republic
[2] University of Miami, USA

Abstract. This paper describes a combination of several Mizar-based tools (the MPTP translator, XSL style sheets for Mizar), and TPTP-based tools (IDV, AGInT, SystemOnTPTP, GDV) used for visualizing, analyzing, and independent verification of Mizar proofs. The combination delivers to the readers of the Mizar Mathematical Library (MML) an easy, powerful, and almost playful way of exploring the semantics and the structure of the library. The key factors for the relative easiness of having these functionalities are the choice of XML as both internal and external interface of Mizar, and the existence of a TPTP representation of MML articles. This shows the great added value that can be obtained by cooperation of several quite diverse (and quite often separately developed) projects, provided that they are based on the same communication standards.

1 Instead of Reading This Paper

Perhaps the first thing a reader of this paper should do is to play with the functionalities that have been implemented. These functionalities provide an easy, powerful, and almost playful way of exploring the semantics and the structure of the Mizar Mathematical Library (MML) [6]. Select one of the HTML files at `http://www.tptp.org/MizarTPTP/`, e.g., the MML article about the Boolean Properties of Sets, `XBOOLE_1`. This will show the HTML rendition of the article, an extract of which is shown in Figure 1. Provided that Java 1.5 is installed and available to the browser, clicking on the palm tree icon next to a theorem will run the Interactive Derivation Viewer (IDV) [12] applet to display the TPTP form [11] of the Mizar proof tree.[3]

Figure 2 shows the IDV window for the first theorem (`Th1`) in `XBOOLE_1`. The many IDV functionalities available there are described later in this paper, on the other hand, many of them are quite self-explanatory and easy to explore. One of them, which might be particularly interesting to "semantically oriented" users, is the verification functionality. The sequence of interactions is shown in Figure 3.

* Supported by a Marie Curie International Fellowship within the 6^{th} European Community Framework Programme.

[3] See the IDV video – `http://www.cs.miami.edu/~geoff/ResearchProjects/ART/IDVVideo.mov`.

ISBN 978-83-7431-229-5 ISSN 0860–150X

Click the "white tick" ("show verified formulae") icon (it turns green), and then the "hurricane flags" icon on its right-hand side ("verify all formulae"), accept the default EP system [7] as the verification ATP system in the pop-up window, and click the "GO CANES" icon in the pop-up window. Green ticks will start to appear in the IDV window, denoting that the (TPTP form of the) Mizar inference have been verified by the GDV derivation verifier [15, 10], using EP for checking logical consequences. Click the "hurricane flags" again to stop the verifications.

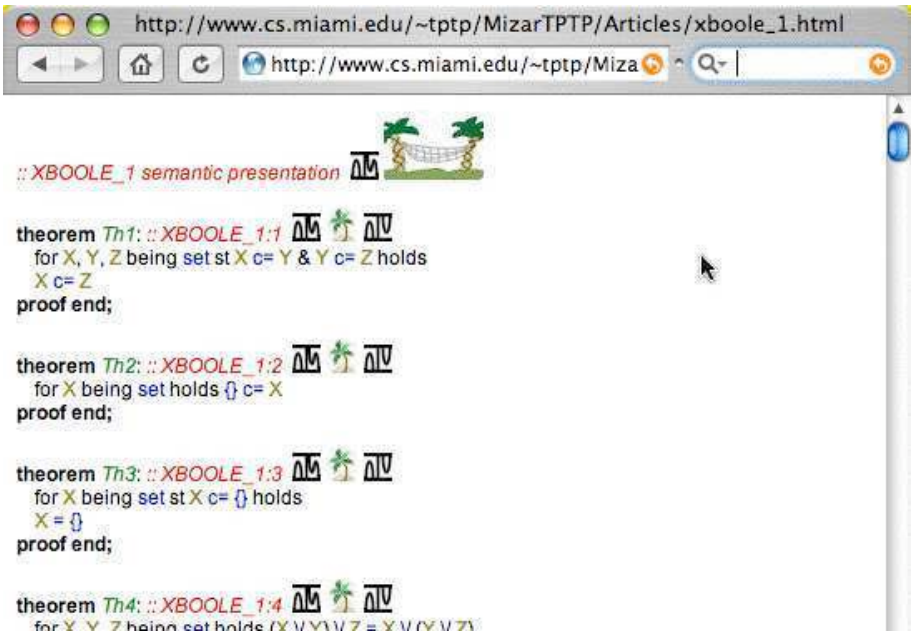


Fig. 1. Article XBOOLE_1

Going back to the HTML presentation and clicking on the "hammock between palm trees" icon will similarly call IDV, now displaying the overall theorem structure of the article. If you think that this is not especially interesting, click this icon in JGRAPH_7 (it will take a while to load the applet), to get the IDV window shown in Figure 4. Would you be able to say just by looking at the HTML (or ASCII) presentation that the Mizar article [3] has this particular derivation structure, and be motivated to explore (and perhaps criticize) the reasons why it is so?

2 Motivation and Overview

There has been quite a lot of work recently on translating the MML to the TPTP format, and on making the TPTP format sufficiently rich for this task. The goal is to make the MML accessible to the automated theorem proving (ATP) systems

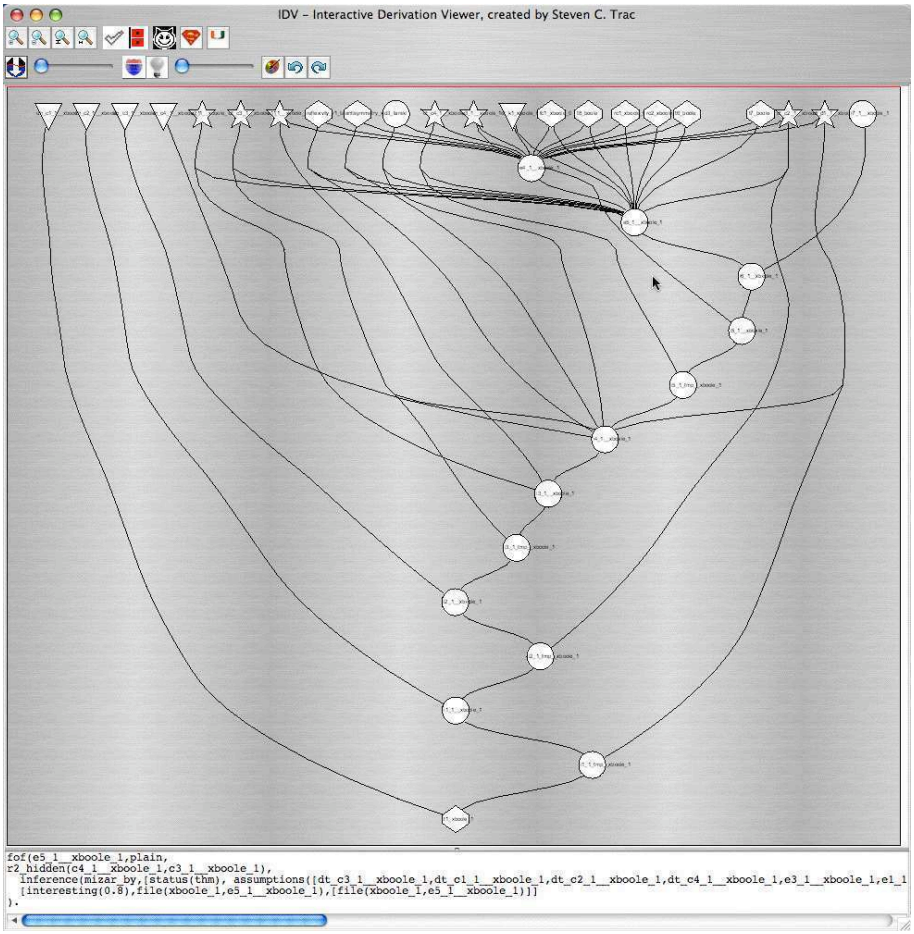


Fig. 2. Theorem `Th1` in Article `XBOOLE_1`

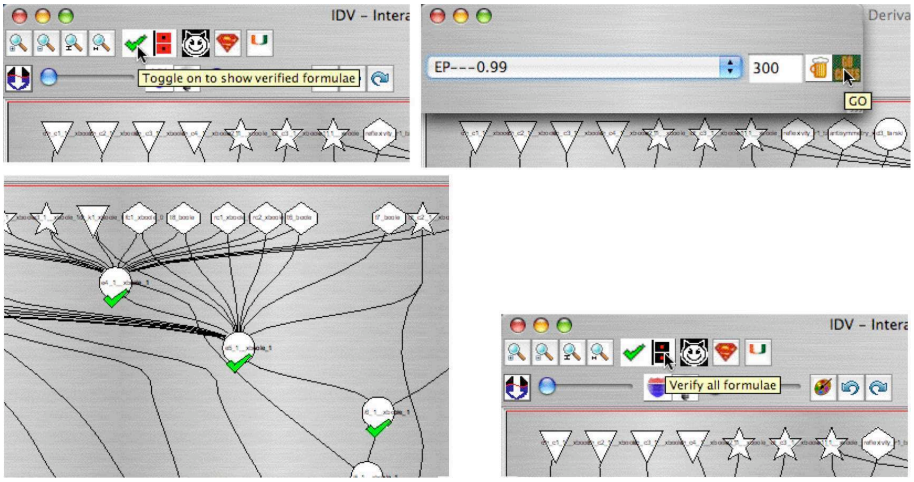


Fig. 3. Verifying `Th1` in Article `XBOOLE_1`

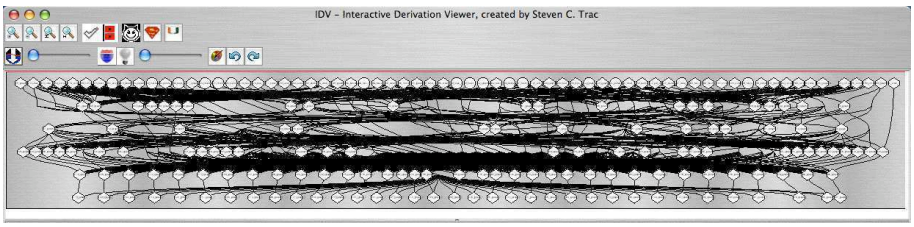


Fig. 4. Article `JGRAPH_7`

that either directly, or through the TPTP translation tools, understand the TPTP language. The systems can then in turn be used for proof assistance over the MML, its independent verification [15], refactoring [17], and many more interesting AI tasks [16]. Similarly, the (XSL-based) HTML presentation of the Mizar library has been continuously developed, with the goal to make it a useful tool for its readers and authors.

There are several other projects aimed at translating large formal corpora to TPTP format, and at reaping the benefits from the unified TPTP interface to ATP systems and tools. Examples include the Isabelle proof assistant, [2], the SUMO ontology, [4], and the Cyc knowledge base [1]. The advantages of developing and using tools that work directly with the TPTP format are obvious. While the SystemOnTPTP interface for solving ATP problems [9] has been well known in the ATP community for a long time, there has also been a significant recent development of tools working with TPTP format derivations. IDV is a tool for graphical presentation of TPTP format derivations, and provides an interface for analysis and verification of derivations. IDV is linked to the AGInT system [5], which assigns interestingness values to derived formulae, based on several AI heuristics. This can be used by IDV to compact large derivations into smaller presentations of the most interesting facts and the links between them. Graphical presentation of a derivation allows a user to quickly get a feel for the structure of the derivation, and interact with the derivation in a more natural way than is possible with a text presentation. Another of IDV's functionalities is its link to the GDV verification system [10]. Recent versions of this system understand a Mizar-like *assumption* extension to the TPTP language, and are capable of independent verification of the Mizar proofs exported to this extended TPTP language by the MPTP system [15].

In short, the work presented here uses the existing (and continuously developed) semantic link between Mizar and TPTP, and capitalizes on that link by re-using the IDV, SystemOnTPTP, AGInT, and GDV systems, for additional semantic presentation and verification purposes. In the following section it is explained how this is (relatively easily) technically done by building on the MPTP system [14] and the XSL style sheets for Mizar [13]. In Section 4 we summarize the new features and improvements of the IDV tool that are used for this (and which, by IDV's nature, are generally available for any derivation in the TPTP format).

3 Structure of the System

The overall structure of the system is shown in Figure 5. The Mizar parser produces the XML form of a Mizar article, which is then translated by XSLT tools to HTML, and by XSL and the MPTP translator to the TPTP format. A number of additions have recently been made to both the XSL translations. First, the original Mizar identifiers (variables, labels, constants) are kept in the XML, and are thus available for more faithful HTML presentation. The presentational information is also retained, which allows re-creation of the original logical connectives used in formulae. This is used for both the HTML and TPTP translations. The TPTP translation has been enhanced to contain all the Mizar natural deduction information necessary for recreating the proof structure. A TPTP format extension

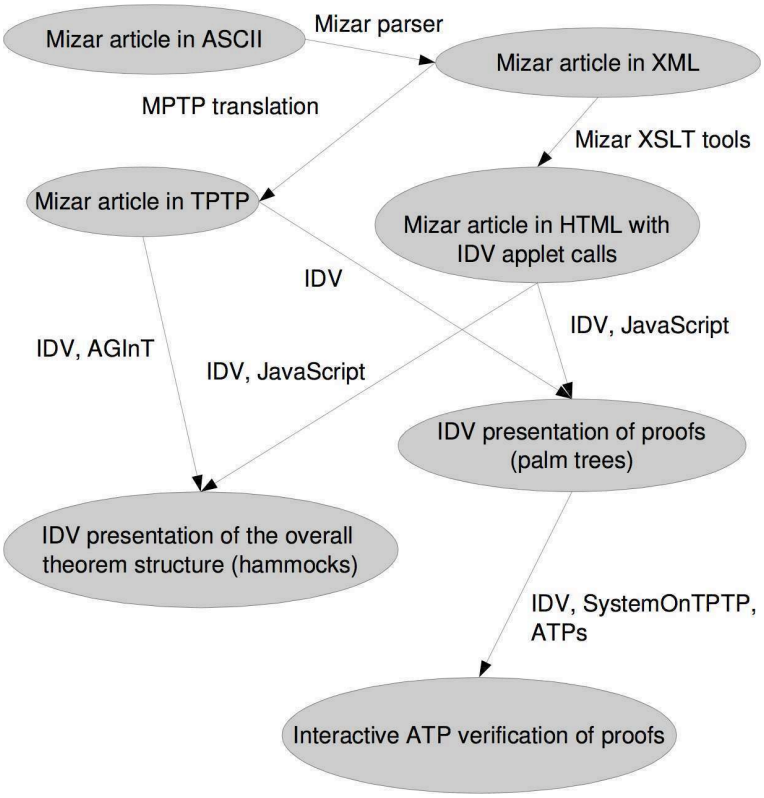


Fig. 5. Systems used for the Semantic Presentation

was implemented for recording proofs that introduce and discharge assumptions, functions that export the Mizar proofs to this format have been written, and the GDV verifier now allows independent verification of such proofs [15].

The linking to the IDV applet and display of the IDV icons in the Mizar HTML (as in Figure 1) is added if the XSL processing uses the `idv` option. This was a simple extension of the existing XSL style sheets, which have gradually become highly parameterized for producing the Mizar HTML in quite different settings. Together with the `idv` option, the `ajax_proofs` option was used. It puts the Mizar format theorem proofs into separate files, and loads and displays them (via an XMLHttpRequest) when the user clicks on the **proof** keyword. This makes the size of the HTML files much smaller, allowing more eye-candy (colors, titles, etc.), and a faster browsing experience. A new `display_thesis` option has also been implemented, which puts a clickable `thesis` text after each Mizar natural deduction step. This is used to display the implicit thesis (computed by Mizar) after each natural deduction step. It is especially useful for this presentation, because the TPTP counterparts of theses are necessary parts of the corresponding TPTP proofs visualized by IDV.

The TPTP format proofs of the theorems in each article are available under the TSTP icon to the right of each theorem header in the HTML presentation (as in Figure 1). The Mizar-to-TPTP translation is also easy to do in real time, and we hope to make this service available in the near future, see Section 6. Note that these proofs are in a format that is intended to be really verifiable by ATP systems. That means that the necessary background information used implicitly by the Mizar proof checker has been added to the problems as axioms. In advanced domains this can make the axiom set quite large, which is unsuitable for direct IDV display. That's one reason why the IDV "red line" functionality for hiding axioms (see Section 4) was developed, and is used for presenting such problems. An interestingness rating was added to each step in each theorem's TPTP format proof, based on the level of nesting the Mizar proof. The "lightbulb" icon and slider in IDV (see Figure 2) allow the user to interactively set an interestingness threshold for the derivation display, and hide nodes whose interestingness is below the threshold, thus displaying a proof synopsis (see Section 4).

The TPTP format problem corresponding to the Mizar problem, as generated by the MPTP system [14], are available under the TPTP icon to the right of the palm tree of each theorem header in the HTML presentation (as in Figure 1). The TPTP problem is an independent translation of the Mizar problem, which can be attempted by any ATP system. Of course the derivation obtained by an ATP system is unlikely to be the same as the TPTP format proof formed by the translation of the Mizar proof of the theorem.

In the same way that individual theorem proofs are translated, the system is used to produce the overall theorem structure of each Mizar article, in the form of summarized TPTP derivations. In these summarized derivations each theorem is a node of the derivation, and its parents are the axioms, definitions, and theorems from which it was proved in Mizar. These are available under the TSTP icon to the right of each article header in the HTML presentation (as in Figure 1). The goal of this presentation is to provide structural information about dependencies between articles' "main results". The example `JGRAPH_7` given in Section 1 shows that the visual information about this high-level structure can be very useful (to the authors, reviewers, or just readers of the Mizar library). This information is intended to be purely presentational, and as such the background information necessary for "high-level" verification is not added. This would actually be very easy to do, but users probably would not like to try to verify these high-level steps because the success rate (in a reasonable time limit for an ATP system) is obviously much lower than for the simple inferences in the individual theorem's proofs. The AGInT system was used on these overall presentations to add an interestingness rating to each theorem, so that IDV can display a synopsis of the overall structure. That again can produce new insights while viewing the high-level derivation structure.

4 Presenting with IDV

IDV is a tool for graphical rendering of derivations that are written in the TPTP format. A number of additions and improvements have recently been done (since [12]) to provide the functionalities needed for the presentation of the Mizar library

and beyond. A description of the features, many of them new, useful for viewing the Mizar proofs is provided here: the summarization, subderivation extraction, and verification functionalities.

4.1 Summarization

The TPTP format proofs, and in particular the article summaries, are very large, and typically have a very high proportion of axioms. Such large derivations are difficult to display in full detail for three reasons. First, IDV runs as a Java applet, which limits its speed. Second, it is hard to see a single formulae node when there are a few thousand of them on the screen. Third, when there is a very high proportion of axioms the display is necessarily very wide because of the axioms lined up across the top, which requires zooming out a great deal to see the whole proof, and the nodes become very small. For derivations that are very large, IDV offers two mechanisms to make the derivation easier to view.

The first mechanism is proof synopsis. As explained in Section 3, an "interestingness" value can be associated with each formulae, either in advance by some external criteria, or by the AGInT system. AGInT may be used in advance (as is done for the Mizar articles), or can be called from within IDV by toggling on the "light bulb" ("show IDV synopsis") icon. When the light bulb is on, nodes are resized proportionally to their interestingness. Moving the interestingness slider to the right increases the interestingness threshold, and nodes with lower interestingness are hidden (with edges being extended from their children to their unhidden ancestors). By default leaf nodes are protected from being hidden, but the new "police badge" ("toggle protection of uninteresting axioms") icon can be used to turn off this protection, thus making it possible to hide large numbers of uninteresting axioms. The "artists palette" ("redraw") icon redraws the derivation with only the displayed nodes, to provide a synopsis of (a part of) the derivation. An example synopsis of the first theorem in `XBOOLE_1` is shown in Figure 6.

The second mechanism is unconditional hiding. The new "diver down" ("hide formulae above the red line") icon and slider allow the user to unconditionally hide formulae above a chosen depth from the axioms. This is particularly useful for (and was motivated by) Mizar article summaries that have a very high proportion of axioms. For example, the overall theorem structure of the article `JGRAPH_7` is slow to display because of the large number of axioms. Using the red line slider to hide the axioms (the top of Figure 7), and then doing a redraw (the bottom of Figure 7), provides a summary of the lower, probably more important, parts of the derivation.

The functionalities described above have been combined to automatically summarize very large derivations that are given to IDV. If there are more than 256 nodes in the derivation, then IDV

1. Adds interestingness (by calling AGInT), unless already supplied by user.
2. Sets the interestingness threshold (i.e., moves the interestingness slider) to try to reduce the number of nodes to less than 256.
3. If more than 256 nodes remain unhidden, sets the axiom protection off.

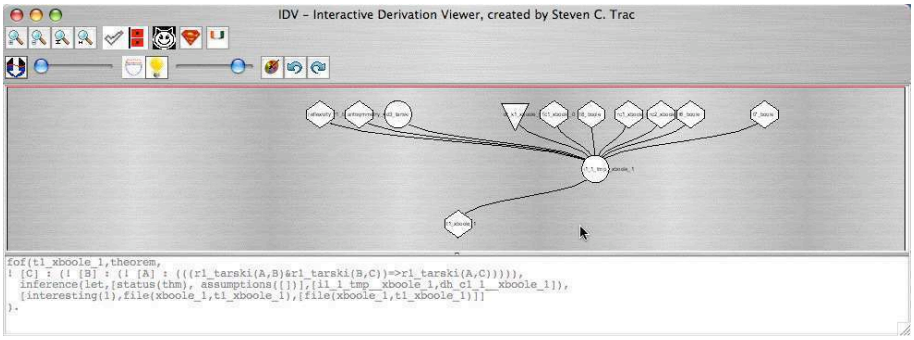


Fig. 6. Synopsis of `Th1` in Article `XBOOLE_1`

4. If more than 256 nodes still remain unhidden, moves the red line down as many levels as necessary.
5. Does a redraw, so that the hidden formulae do not affect the current drawing. The user can move the sliders back and toggle/untoggle buttons to show hidden formulae later.

This automatic summarization can be seen, e.g., in the display of the article `JGRAPH_4`.

Together with an optimization of the AGInT system on very large data (thousands of derivation steps are now rated within seconds), these mechanisms have largely sped up and improved the display of the Mizar proofs.

4.2 Subderivation Extraction

There are now more ways to interact with the IDV graph, making it easier to explore the proof in different ways, and render selected extracts.

The first group of extracts are determined by mouse clicks on a chosen node. A left mouse click opens a pop-up window with the text of the annotated formula and its parents. This window also allows verification of that inference, as explained in Section 4.3. Note that the parents shown are of the "what you see is what you get" nature, where hidden ones from non-interestingness and the red line are not shown.

A control left mouse click on a node opens a new IDV window showing the formula and its parents, with the full functionality of any IDV window. A shift-control left mouse click opens a new IDV window showing the formula and all its ancestors. This is useful for extracting a subderivation - Figure 8 shows the subderivation rooted at the node `e5_1__xboole_1` of the first theorem in `XBOOLE_1`. A control right mouse click opens a new IDV window showing the formula and its immediate descendants and their parents. A shift-control right mouse click opens a new IDV window showing the formula and all its descendants and their parents. This is useful for seeing what formulae depend on the clicked one - Figure 9 shows the descendant derivation hung from the node `t7_jgraph_7` of the `JGRAPH_7` article.

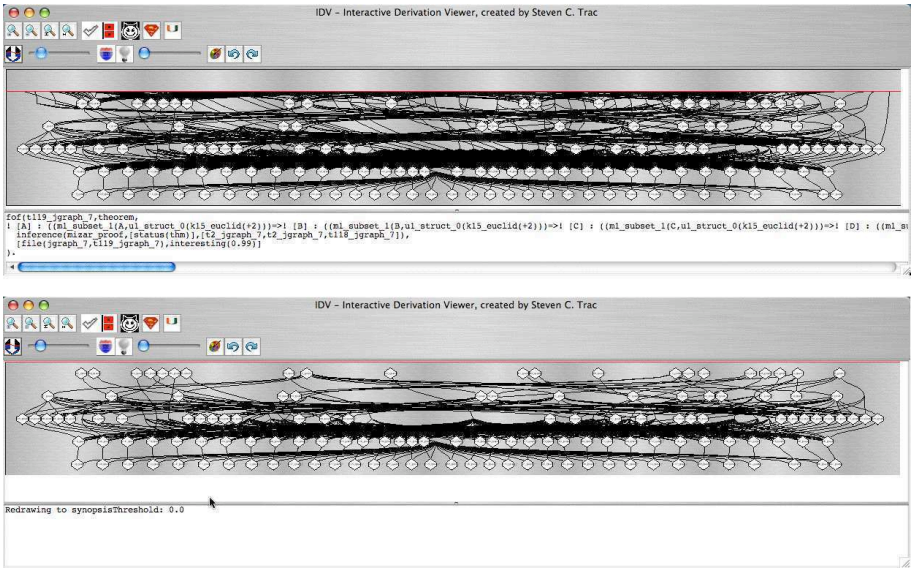


Fig. 7. Summary of Article `JGRAPH_7`

The new "good/evil cat" ("show only axioms and lemmas") icon hides nodes that are not axioms or logical consequences of the axioms. In proofs by contradiction, which negate the conjecture, this hides formulae that are derived from the negated conjecture. In proofs that have assumptions, this hides the assumptions and any formulae that have undischarged assumptions. This is useful for identifying formulae that can be used, e.g., as lemmas from the axioms.

4.3 Verification

A derivation displayed by IDV can be verified in three ways. Regardless of how the verification is done, if a node has been verified and the "green tick" ("show verified formulae") icon is on, verified nodes will have a green tick on them (see Figure 3). The verified status is reset if proof summarization changes the apparent parents of the node.

The first way to verify nodes is as illustrated in Section 1. IDV iteratively calls either GDV or a chosen ATP system to verify each node of the derivation. The choice of whether to use GDV or an ATP system is controlled by the "beer" – GDV – and "cocktail" – ATP system – icon in the pop-up window shown in the upper right of Figure 3. If an ATP system is used directly then only steps of logical consequence are verified, by proving them using the ATP system. If GDV is used then all inference steps are verified, proving steps of logical consequence using the ATP system, and using other techniques in other situations. In particular, GDV is able to verify the propagation and discharge of assumptions.

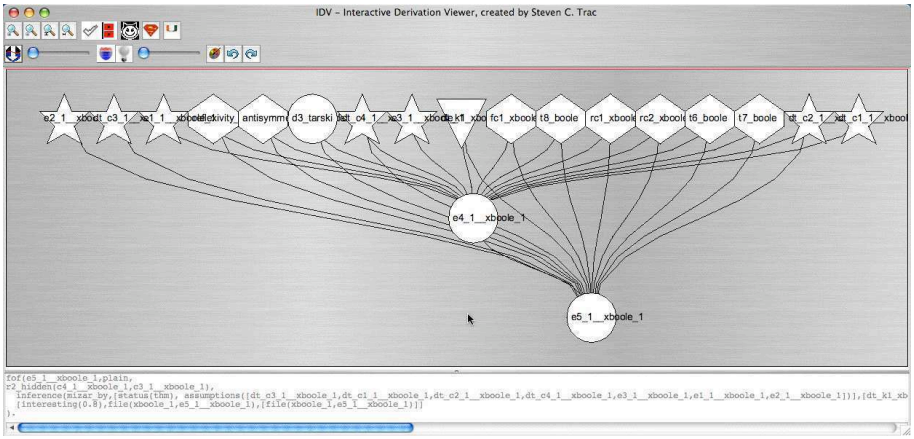


Fig. 8. Extract from `Th1` in Article `XBOOLE_1`

The second way is to verify an individual node from it's pop-up window, produced by clicking on the node (see Section 4.2). Again, there is a choice of using GDV or an ATP system. If an ATP system is used and a TPTP format proof is returned by the ATP system, the "palm tree" ("new IDV window") icon will open a new IDV window displaying the verifying ATP system's proof. Figure 10 show's EP 0.99's verification proof of the final node `t1_xboole_1` of the first theorem in `XBOOLE_1`.

The third way is to use the "superman" ("SystemOnTSTP") icon, which exports the derivation to the SystemOnTSTP interface[4], which in turn provides access to a range of derivation analysis and display tools. The GDV tool is available there for a complete verification of the derivation, including structural checks that are not done from within IDV. This interface is shown in Figure 11.

5 Conclusion

This paper describes a combination of Mizar- and TPTP-based tools used for visualizing, analyzing, and independent verification of Mizar proofs. The combination delivers to the readers of the MML an easy, powerful, and almost playful way of exploring the semantics and the structure of the library. The key factors for the relative easiness of having these functionalities are the choice of XML as both internal and external interface of Mizar, and the existence of a TPTP representation of MML articles.

The system integrates so many components that it naturally behaves as a large debugger for the various tools[5]. This has resulted in battle hardening of the tools,

[4] `http://www.tptp.org/cgi-bin/SystemOnTSTP`

[5] Just a recent example: While randomly inspecting the large number of Mizar derivations in IDV, it has turned out that some cannot be verified, because of a recently introduced

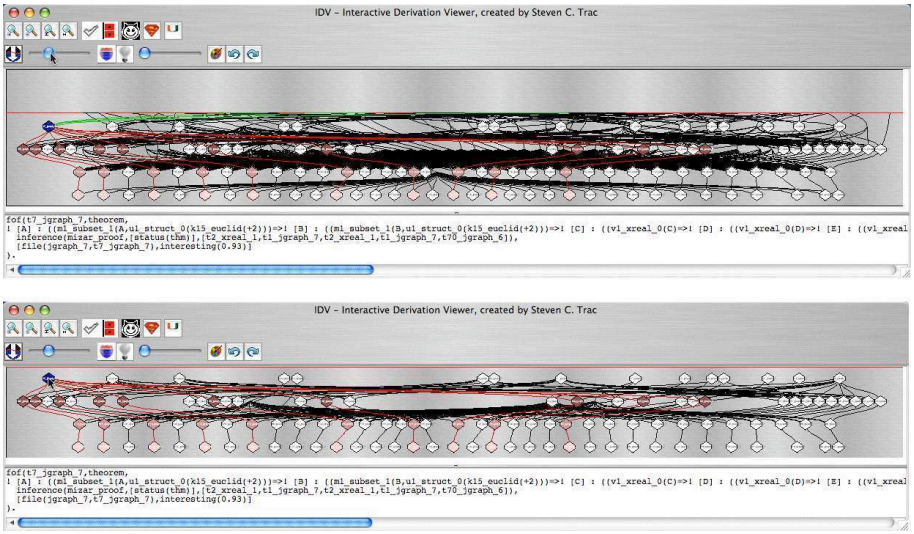


Fig. 9. Extract from Article `JGRAPH_7`

and a robust and reliable interface. The combination shows the great added value that can be obtained by cooperation of several quite diverse (and, quite often, separately developed) projects, provided that they are based on the same communication standards. This places the system alongside other work based around a combination of component reasoning systems, e.g., [19, 18, 8].

Although much of work done was motivated by the desire to view the structure of Mizar proofs and articles, all of the work is general and immediately available for any derivations in the TPTP format. As such all the tools are now part of the general SystemOnTSTP interface.

6 Future Work

Obvious future work is to make the systems presented here available in a dynamic way, so that Mizar authors can use the functionalities described here on newly written articles. This would correspond to the existing dynamic functionalities of the SystemOnTPTP, used for interactive online work with ATP problems and derivations.

An initial implementation of such a system has been started and is available for experiments at `http://octopi.mizar.org/~mptp/MizAR.html`. This web interface now provides the possibility to verify an article using Mizar, present it in HTML linked to the full Mizar library, use MPTP to translate the article to TPTP, verify simple justifications using ATPs, and present such ATP proofs using IDV.

incompatibility between the TPTP Java parser used by IDV, and the TPTP parser used by the E prover.

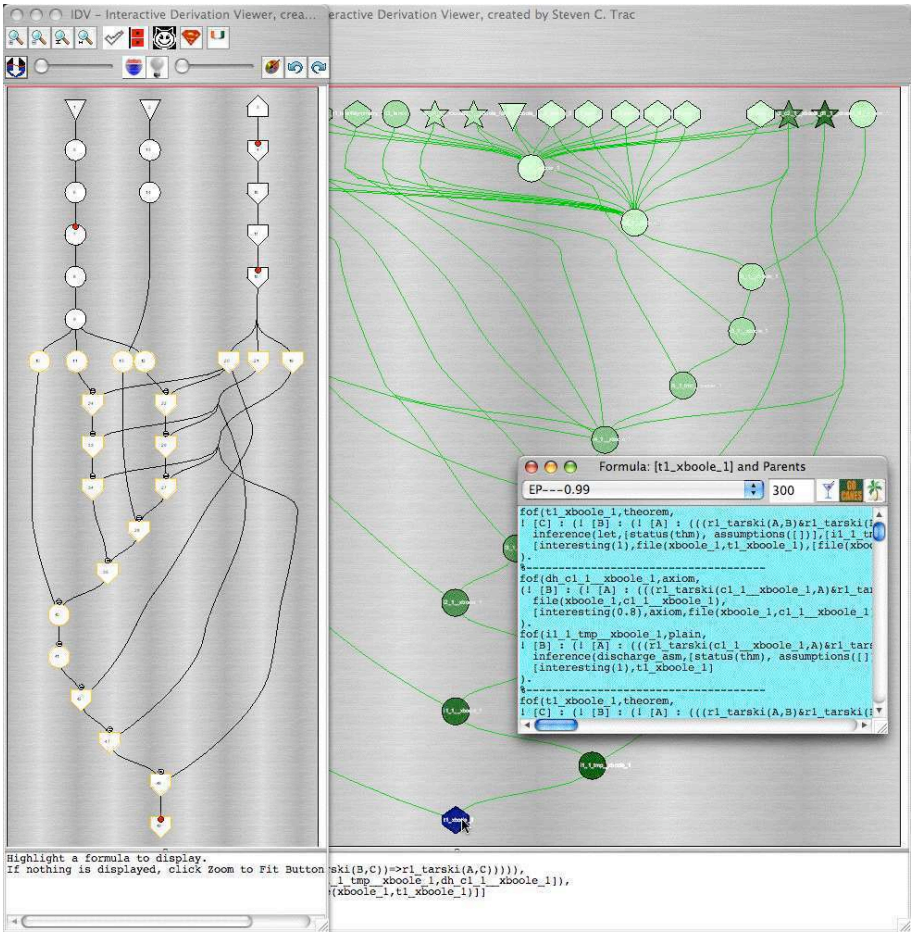


Fig. 10. Node Verification from `Th1` in Article `XBOOLE_1`

The initial goal of this web interface is to provide Mizar authors with detailed explanation of the Mizar simple justifications, however many other presentation and verification features described above can already be added quite easily. Another very simple addition would be wiki-like functionalities: creating a semantically disambiguated XML document and linked HTML document from an ASCII Mizar article involves just running Mizar and XSLT processor, so the only missing feature for a basic wiki is just some version system (e.g. RCS), and possibly user tracking.

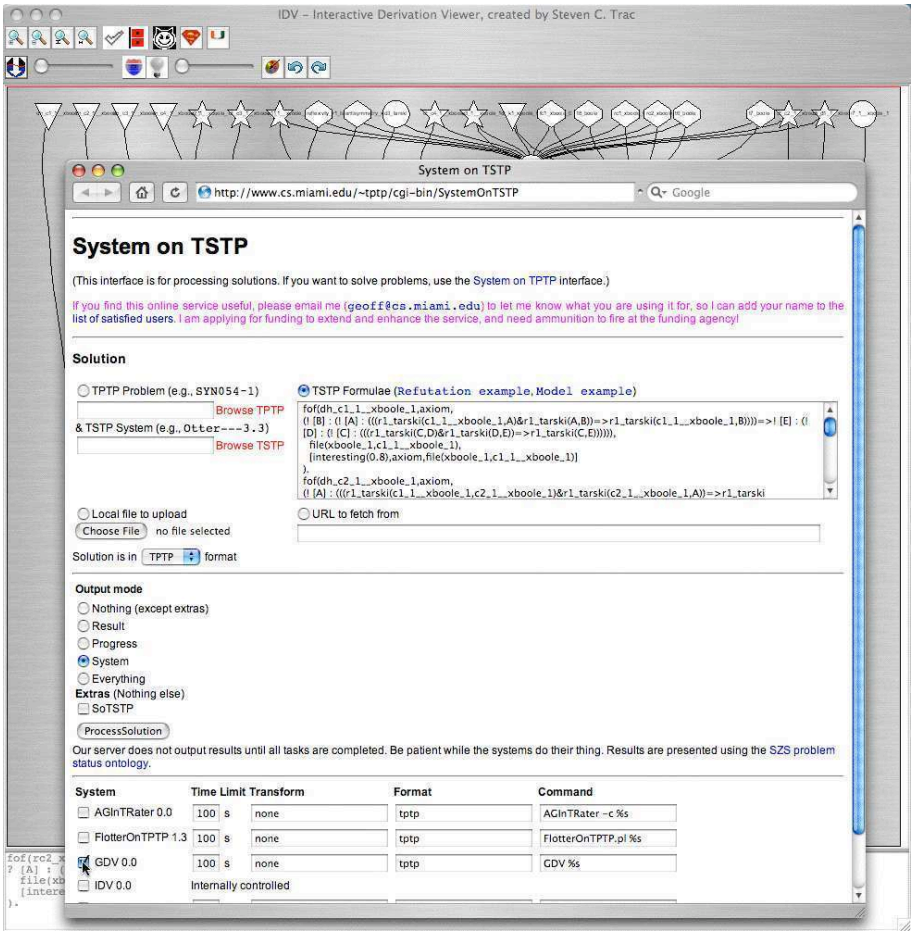


Fig. 11. SystemOnTSTP for `Th1` in Article `XBOOLE_1`

References

1. Matuszek, C., Cabral, J., Witbrock, M., and DeOliveira, J.: An Introduction to the Syntax and Content of Cyc. In Baral C., editor, *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pp. 44–49, 2006.
2. Meng, J. and Paulson, L.: Translating Higher-Order Problems to First-Order Clauses. In G. Sutcliffe, R. Schmidt, and S. Schulz, editors, *Proceedings of the FLoC'06 Workshop on Empirically Successful Computerized Reasoning, 3rd International Joint Conference on Automated Reasoning*, volume 192 of *CEUR Workshop Proceedings*, pp. 70–80, 2006.
3. Nakamura, Y. and Trybulec, A.: The Fashoda Meet Theorem for Rectangles. *Formalized Mathematics*, 13(2), pp. 199–219, 2005.

4. Niles, I. and Pease, A.: Towards A Standard Upper Ontology. In C. Welty and B. Smith, editors, *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems*, pp. 2–9, 2001.
5. Puzis, Y., Gao, Y., and Sutcliffe, G.: Automated Generation of Interesting Theorems. In G. Sutcliffe and R. Goebel, editors, *Proceedings of the 19th International FLAIRS Conference*, pp. 49–54. AAAI Press, 2006.
6. Rudnicki, P.: An Overview of the Mizar Project. In *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, pp. 311–332, 1992.
7. Schulz, S.: A Comparison of Different Techniques for Grounding Near-Propositional CNF Formulae. In S. Haller and G. Simmons, editors, *Proceedings of the 15th International FLAIRS Conference*, pp. 72–76. AAAI Press, 2002.
8. Sorge, V., Meier, A., McCasland, R., and Colton, S.: Automatic Construction and Verification of Isotopy Invariants. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pp. 36–51. Springer-Verlag, 2006.
9. Sutcliffe, G.: SystemOnTPTP. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, number 1831 in Lecture Notes in Artificial Intelligence, pp. 406–410. Springer-Verlag, 2000.
10. Sutcliffe, G.: Semantic Derivation Verification. *International Journal on Artificial Intelligence Tools*, 15(6), pp. 1053–1070, 2006.
11. Sutcliffe G., Schulz, S., Claessen, K., and Van Gelder, A.: Using the TPTP Language for Writing Derivations and Finite Interpretations. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pp. 67–81, 2006.
12. Trac, S., Puzis, Y., and Sutcliffe, G.: An Interactive Derivation Viewer. In S. Autexier and C. Benzmüller, editors, *Proceedings of the 7th Workshop on User Interfaces for Theorem Provers, 3rd International Joint Conference on Automated Reasoning*, volume 174 of *Electronic Notes in Theoretical Computer Science*, pp. 109–123, 2006.
13. Urban, J.: XML-izing Mizar: Making Semantic Processing and Presentaion of MML Easy. In M. Kohlhase, editor, *Proceedings of the 4th International Conference on Mathematical Knowledge Management*, volume 3863 of *Lecture Notes in Computer Science*, pp. 346–360, 2005.
14. Urban, J.: MPTP 0.2: Design, Implementation, and Initial Experiments. *Journal of Automated Reasoning*, 37(1-2), pp. 21–43, 2006.
15. Urban, J. and Sutcliffe, G.: ATP Cross-verification of the Mizar MPTP Challenge Problems. In N. Dershowitz and A. Voronkov, editors, *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 4790 in Lecture Notes in Artificial Intelligence, pp. 546–560, 2007.
16. Urban, J., Sutcliffe, G., Pudlak, P., and Vyskocil, J.: MaLARea SG1: Machine Learner for Automated Reasoning with Semantic Guidance. In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pp. 441–456, 2008.
17. Urban, J.: MoMM – fast interreduction and retrieval in large libraries of formalized mathematics. *International Journal on Artificial Intelligence Tools*, 15(1), pp. 109–130, 2006.
18. Zimmer, J. and Autexier, S.: The MathServe System for Semantic Web Reasoning Services. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pp. 17–20, 2006.

19. Zimmer, J., Meier, A., Sutcliffe, G., and Zhang, Y.: Integrated Proof Transformation Services. In C. Benzmüller and W. Windsteiger, editors, *Proceedings of the Workshop on Computer-Supported Mathematical Theory Development, 2nd International Joint Conference on Automated Reasoning*, Electronic Notes in Theoretical Computer Science, 2004.

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 18 (31) 2009

Statistics on Digital Libraries of Mathematics

Freek Wiedijk

Institute for Computing and Information Sciences
Radboud University Nijmegen
Toernooiveld 1, 6525 ED Nijmegen, The Netherlands

Abstract. We present statistics on the standard libraries of four major proof assistants for mathematics: HOL Light, Isabelle/HOL, Coq and Mizar.

1 Introduction

1.1 Problem

The advent of digital computers has introduced a new way of doing mathematics called 'formalized mathematics'. In this style of doing mathematics one encodes the mathematics in the computer in sufficient detail that the computer can fully check the correctness according to a small number of logical rules. This style of doing mathematics is much more precise and trustable than the traditional way of first understanding the mathematics in one's head and then just writing it on a blackboard or on paper. Also it is a very pleasurable experience to write down one's mathematics in a way that *all* the details are there, knowing that there is nothing left implicit.

However, these positive aspects of formalized mathematics have to be paid for. Generally it takes much longer to turn mathematics into formalized form than it takes to just understand it, or even than to write it down in a traditional way. (A rough estimate might be that it takes about ten times as long to formalize something than it takes to write it down in meticulous traditional 'textbook style'.) One might wonder where this time is going, i.e., how much it is spent on the various aspects of formalization. For instance there are the aspects of formalizing the definitions, choosing good notation for the defined notions, then stating the appropriate formal statements to be proved, and finally writing the formal proofs themselves.

Another question that might be posed is whether there are significant differences in the time needed for these activities between the different *systems* for formalization of mathematics.

In this paper we will study these questions. We will not do this by focusing on the *activity* of formalization, but rather by studying the *results* of this activity, the libraries of formalized mathematics that have been created by the various research communities that work on this subject. These libraries have grown into quite large human 'artifacts', which – we claim – deserve study in their own right. In this

ISBN 978-83-7431-229-5 ISSN 0860–150X

paper we will do this by collecting various statistics on these libraries. One might compare our work here to that of a biologist who just makes an inventarization of the different species that are out there in the world. In this paper we mainly just collect data.

The question that we will address here is what are the different aspects of formalization that one can find in the formalized libraries that are out there, how much of those libraries is spent on which of these aspects, and whether the different systems for formalization are more or less similar in these aspects or whether they have significant differences.

1.2 Approach

The way that we count the libraries of formalized mathematics is as follows. First we concatenate all the files for a system into one huge file. Then we tag each line of this file with the *category* of that line, and then we count the different types of lines that we find.

We will explain this procedure with a small example. Here is a very small formalization of the irrationality of the square root of two by John Harrison in the HOL Light system (taken from *The Seventeen Provers of the World* [5], a collection of formalizations of this irrationality proof in various systems):

```
(* ------------------------------------------------------------------------- *)
(* Definition of rationality (& = natural injection N->R).                   *)
(* ------------------------------------------------------------------------- *)

let rational = new_definition
  `rational(r) = ?p q. ~(q = 0) /\ abs(r) = &p / &q`;;

(* ------------------------------------------------------------------------- *)
(* The main lemma, purely in terms of natural numbers.                       *)
(* ------------------------------------------------------------------------- *)

let NSQRT_2 = prove
 (`!p q. p * p = 2 * q * q ==> q = 0`,
  MATCH_MP_TAC num_WF THEN REWRITE_TAC[RIGHT_IMP_FORALL_THM] THEN
  REPEAT STRIP_TAC THEN FIRST_ASSUM(MP_TAC o AP_TERM `EVEN`) THEN
  REWRITE_TAC[EVEN_MULT; ARITH] THEN REWRITE_TAC[EVEN_EXISTS] THEN
  DISCH_THEN(X_CHOOSE_THEN `m:num` SUBST_ALL_TAC) THEN
  FIRST_X_ASSUM(MP_TAC o SPECL [`q:num`; `m:num`]) THEN
  POP_ASSUM MP_TAC THEN CONV_TAC SOS_RULE);;

(* ------------------------------------------------------------------------- *)
(* Hence the irrationality of sqrt(2).                                       *)
(* ------------------------------------------------------------------------- *)

let SQRT_2_IRRATIONAL = prove
 (`~rational(sqrt(&2))`,
  SIMP_TAC[rational; real_abs; SQRT_POS_LE; REAL_POS; NOT_EXISTS_THM] THEN
  REPEAT GEN_TAC THEN DISCH_THEN(CONJUNCTS_THEN2 ASSUME_TAC MP_TAC) THEN
  DISCH_THEN(MP_TAC o AP_TERM `\x. x pow 2`) THEN
  ASM_SIMP_TAC[SQRT_POW_2; REAL_POS; REAL_POW_DIV; REAL_POW_2; REAL_LT_SQUARE;
               REAL_OF_NUM_EQ; REAL_EQ_RDIV_EQ] THEN
  ASM_MESON_TAC[NSQRT_2; REAL_OF_NUM_EQ; REAL_OF_NUM_MUL]);;
```

Now for each system studied in this paper, which includes the HOL Light system, we wrote a small perl script that tags each line in a formalization with its category. In our investigation we applied it to the full HOL Light library, but this is what we get when we tag just this example formalization with it:

```
C (* ------------------------------------------------------------------------- *)
C (* Definition of rationality (& = natural injection N->R).                   *)
C (* ------------------------------------------------------------------------- *)
B
D let rational = new_definition
D   `rational(r) = ?p q. ~(q = 0) /\ abs(r) = &p / &q`;;
B
C (* ------------------------------------------------------------------------- *)
C (* The main lemma, purely in terms of natural numbers.                       *)
C (* ------------------------------------------------------------------------- *)
B
T let NSQRT_2 = prove
T  (`!p q. p * p = 2 * q * q ==> q = 0`,
P   MATCH_MP_TAC num_WF THEN REWRITE_TAC[RIGHT_IMP_FORALL_THM] THEN
P   REPEAT STRIP_TAC THEN FIRST_ASSUM(MP_TAC o AP_TERM `EVEN`) THEN
P   REWRITE_TAC[EVEN_MULT; ARITH] THEN REWRITE_TAC[EVEN_EXISTS] THEN
P   DISCH_THEN(X_CHOOSE_THEN `m:num` SUBST_ALL_TAC) THEN
P   FIRST_X_ASSUM(MP_TAC o SPECL [`q:num`; `m:num`]) THEN
P   POP_ASSUM MP_TAC THEN CONV_TAC SOS_RULE);;
B
C (* ------------------------------------------------------------------------- *)
C (* Hence the irrationality of sqrt(2).                                       *)
C (* ------------------------------------------------------------------------- *)
B
T let SQRT_2_IRRATIONAL = prove
T  (``~rational(sqrt(&2))`,
P   SIMP_TAC[rational; real_abs; SQRT_POS_LE; REAL_POS; NOT_EXISTS_THM] THEN
P   REPEAT GEN_TAC THEN DISCH_THEN(CONJUNCTS_THEN2 ASSUME_TAC MP_TAC) THEN
P   DISCH_THEN(MP_TAC o AP_TERM `\x. x pow 2`) THEN
P   ASM_SIMP_TAC[SQRT_POW_2; REAL_POS; REAL_POW_DIV; REAL_POW_2; REAL_LT_SQUARE;
P                REAL_OF_NUM_EQ; REAL_EQ_RDIV_EQ] THEN
P   ASM_MESON_TAC[NSQRT_2; REAL_OF_NUM_EQ; REAL_OF_NUM_MUL]);;
B
```

The lines with a '`C`' are comment lines, the ones with a '`B`' are blank, and so on. The perl script is *ad hoc* in the sense that occasionally it will tag a line wrong. However, by inspecting its output we improved it until it was sufficiently good for our purposes. We claim that our perl scripts will tag less than 1% of the lines in a formalization with the wrong tag.

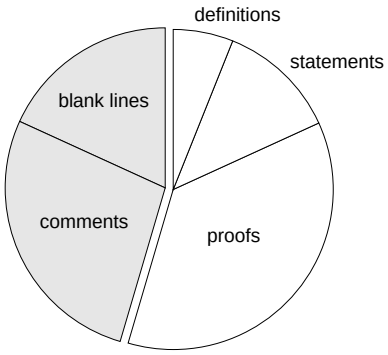
Finally we count the lines in this 'tagged' file for the various tags, and present the results in tabular format. For this small example that table then becomes:

		lines	*bytes*	
`B`	blank lines	6	65	18 %
`C`	comment lines	9	720	27 %
`D`	definitions	2	85	6 %
`T`	theorem statements	4	114	12 %
`P`	proof lines	12	746	36 %
	total	33	1,730	100 %

Of course for a very small example like this, the percentages are not very meaningful. For instance, the number of blank and comment lines is quite a bit higher than it is in a more extensive HOL Light formalization.

The percentages in this table are in terms of line counts, and not in terms of bytes. We believe that line counts is the more interesting measure. It indicates how much one can oversee behind a computer screen without scrolling. (This means that a formalization style where multiple steps are put together on a single line – a formalization style that both John Harrison and Georges Gonthier use – is superior to a more 'programming' like style in which each step gets a line of its own. Georges Gonthier convinced us in a personal communication that line counts is the better way to measure formalizations.)

Finally, we also present the table as a pie chart, as a graphical summary of the results. For these charts the different kinds of lines are grouped together into only seven categories. (In the example two of these categories are missing.) The pie chart for the example is:



Example

1.3 Related work

We are not aware of already existing research into the statistics of formalizations.

In the field of programming, counting lines of source code is one of the methods in the subject called *software metrics*. However, there generally the focus is not on the different *kinds* of source code lines and their function in the programming languages, but more on programmer productivity.

1.4 Contribution

The investigation presented here is a snapshot in time. Also there is not too much 'depth' in our results: really all we did was count. However getting the software that tagged the formalizations reasonably accurate took quite an effort, so obtaining the numbers in this paper was significant work.

The main value of the research described in this paper is showing that all systems for formalizations are quite similar despite their large differences both in foundations and in interaction styles.

The observations in this paper might be a guide for people who design systems for formalizations, by pointing out from the start which elements will need to be part of their formalization language. That way, these elements can all be designed in from the start and will not have to appear as an afterthought later.

Finally this paper can be used as a guide for people who are interested in formalization of mathematics and want to get an impression of the current state of some important libraries.

1.5 Outline

The structure of this paper is as follows. In Section 2 we present an overview of our results. Then in Section 3 we give the statistics in detail. In Section 4 we discuss the different types of lines that we distinguished, and relate them between systems. Finally, in Section 5 we draw some conclusions from our data and indicate possible future work.

2 Overview

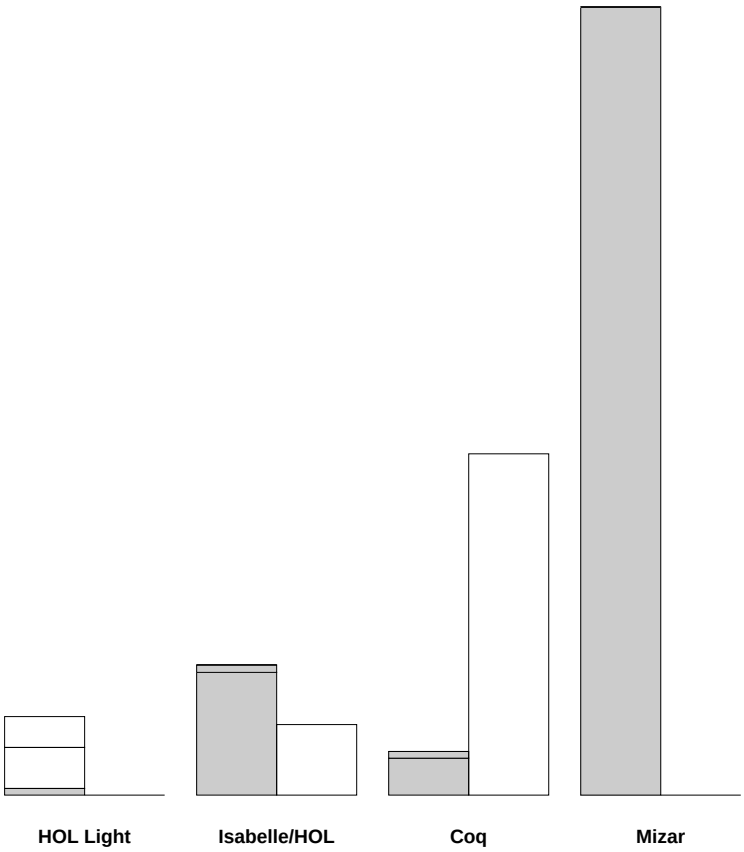
The four systems that we selected for this investigation were:

- HOL Light [1]
- Isabelle/HOL [3]
- Coq [4]
- Mizar [2]

Other systems that we considered were HOL4, ProofPower and PVS. The first two are rather similar to HOL Light, so we did expect them to get quite similar statistics. In the HOL family of systems HOL Light is the system that has been used most for formalization of mathematics. For this reason we selected HOL Light from that group, and left the other two systems out.

The PVS system is one of the most popular systems for formal methods in computer science. It has a very interesting way of dealing with partial functions (called 'predicate sub-types'), and it has strong automation. However it has not been used as much for formalization of mathematics as the other systems that we looked at. For this reason we left PVS out of our comparison as well.

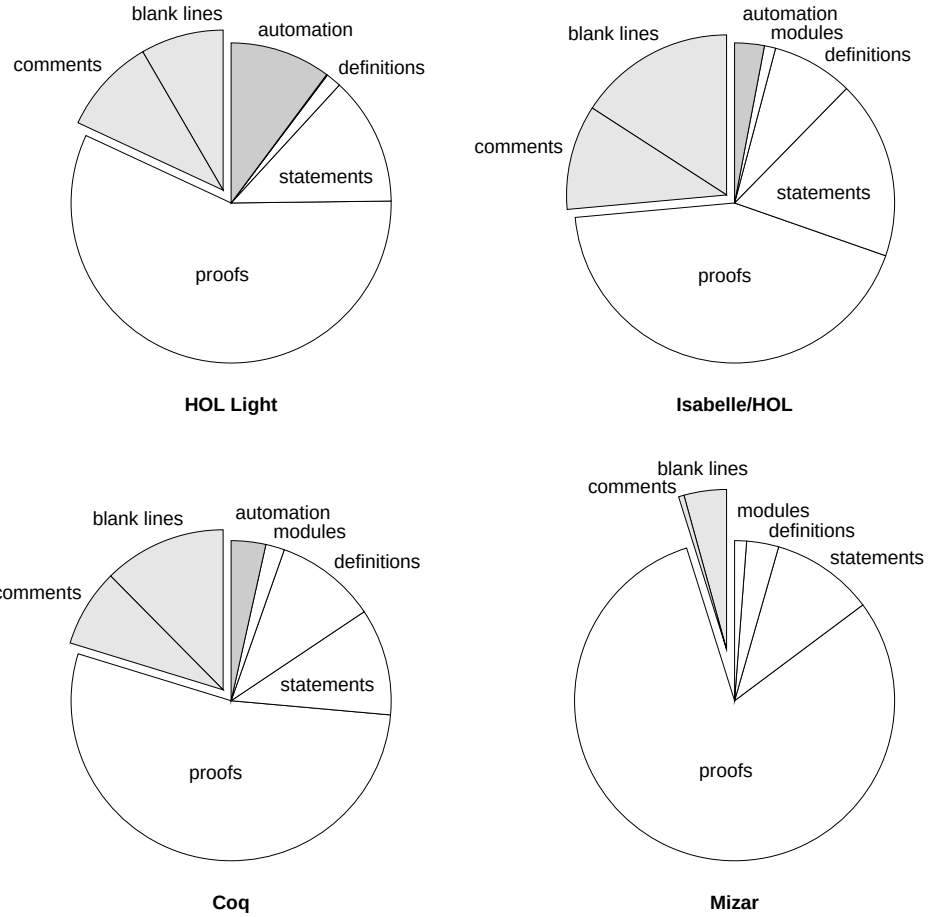
The four systems that we study here have mathematical libraries of quite different sizes. The sizes of these libraries is shown in the bar diagram on the next page. In this diagram the left bars represent the libraries that are distributed with the system. If you install the system, you will have these source files as part of the distribution.



In these bars the gray parts are the parts of the library that can be considered to be the library of the 'core' system. It is that part of the library of the system that is available without doing anything special. In the bar for the HOL Light system the rest of the library (the white part of the bar) has been divided into two sub-parts: the part written by John Harrison, and the part written by other people (which is mostly the formalization of the Jordan Curve Theorem by Tom Hales.) In the gray bar of the Isabelle system the small part at the top corresponds to the sub-directory of the contribution called '`HOLCF`', while in the Coq system it corresponds to the sub-directory '`contrib`'.

The right bars represent libraries that are distributed separately from the system itself. These are collections of formalizations by users of the system that have not (yet) been integrated into the standard library of the system itself. In the case of Coq this is called the Coq *contribs* (short for 'contributions'), while in the Isabelle community it is called the AFP (Archive of Formal Proofs). The Mizar system also has a library of user formalizations called MML (= Mizar Mathematical Library), but in the case of Mizar those formalizations *are* integrated into the standard library of the system.

We now show a summary of the statistics from this paper in the shape of four pie charts:



For most people the bar chart on the previous page and these pie charts will be the most interesting part of this paper.

In the HOL Light pie chart there is a tiny sliver between 'automation' and 'definitions' for the very few lines related to 'modules', but it was too narrow to be labeled. In the Mizar pie chart the 'registration' lines have been included in the 'statements' part, although we gave them category 'H' (which in Section 4 is in the sub-section about automation; we will discuss this there in more detail.)

3 Line Counts

We now present the detailed statistics of the four systems, and discuss which files were counted and which were not.

3.1 HOL Light

The statistics in this paper are on version 2.20 of the HOL Light system.

HOL Light input files have suffix '`.ml`'. These files both contain the implementation of the system as well as the mathematical library, all mixed together. We divided the files that were primarily implementing the system from files that were primarily proving the library in the following way:

The basic HOL Light system consists of a file called `make.ml`, which loads the main file `hol.ml`, which then loads 44 more `.ml` files. (Apart from these 46 files there are in the top level directory 14 more `.ml` files with names of the form '`pa_j_`... `.ml`' related to the input processing of the `ocaml` system that reads the HOL Light files.)

Now the `hol.ml` file is divided into sections. One of these sections has the header 'Mathematical theories and additional proof tools.' We decided that the 19 files in that section were the 'mathematical library' while the 25 files in the other six sections contained the 'implementation of the system'.

Apart from these 19 files we also counted the 'auxiliary library' consisting of 169 `.ml` files in 11 sub-directories. (There were 11 more `.ml` files in another sub-directory named `Proofrecording`. However that is an alternative implementation of the core system, and therefore was left out of this investigation.)

Altogether the number of files counted were:

19 files	*from* `*.ml`
169 files	`*/*.ml`
188 files	

And the statistics about these files were:

		lines	*bytes*	
`B`	blank lines	16,438	21,371	8.4 %
`C`	comments	19,044	864,913	9.7 %
`S`	imports	182	5,459	0.1 %
`D`	definitions	2,547	107,835	1.3 %
`N`	interfaces	486	19,525	0.2 %
`X`	automation: program code	19,979	833,040	10.2 %
`T`	theorem statements	25,493	1,073,208	13.0 %
`P`	proof lines	112,088	4,356,332	57.1 %
	total	196,257	7,281,683	100.0 %

3.2 Isabelle/HOL

The statistics in this paper are on the Isabelle2007 version of the Isabelle/HOL system.

Isabelle has two kinds of files, with suffixes '`.ML`' and '`.thy`'. We decided that the `.ML` files primarily contained the implementation of the system, while the `.thy`

files primarily contained the mathematical library. Now the Isabelle system can be used with different *logics*. The HOL logic is the dominant logic that is used by almost all Isabelle users. For this reason we counted the `.thy` files inside the `HOL` directory (together with `HOLCF` directory, which is closely related). However, we left out the `HOL/Import` sub-directory as it does not contain mathematics but is about importing theories from other systems.

The number of files counted were (here `**` stands for zero or more sub-directory levels in between):

649 files	*most of* `src/HOL/**/*.thy`
88 files	`src/HOLCF/**/*.thy`
737 files	

And the statistics about these files were:

		lines	*bytes*	
`B`	blank lines	51,610	80,304	15.9 %
`C`	source comments	18,941	829,132	5.8 %
`E`	document markup	15,488	713,503	4.8 %
`S`	imports & sectioning	2,838	43,584	0.9 %
`L`	locales	1,394	58,862	0.4 %
`D`	definitions	23,386	1,165,813	7.2 %
`N`	notation	2,736	129,089	0.8 %
`H`	automation: directives	4,022	191,544	1.2 %
`X`	automation: program code	5,714	225,786	1.8 %
`T`	theorem statements	58,659	3,136,976	18.0 %
`P`	proof lines	140,596	5,024,717	43.2 %
	total	325,384	11,599,310	100.0 %

3.3 Coq

The statistics in this paper are about version 8.1 of the Coq system.

Coq files have the suffix '`.v`'. (The implementation of the system is in files with suffixes '`.mli`' and '`.ml`', but unlike HOL Light and Isabelle these are in a different part of the distribution, and are not mixed together with the mathematical library.)

The main library is in the sub-directory `theories`. There is a supplementary library in the sub-directory `contrib`, which mostly contains the supporting theory for several automated proof procedures. (In this second directory the `.v` files and the `.mli` and `.ml` files *are* together. However we also only looked at the `.v` files there.)

Altogether the number of files counted were:

252 files	`theories/*/*.v`
57 files	`contrib/*/*.v`
309 files	

And the statistics about these files were:

		lines	*bytes*	
B	blank lines	13,531	22,456	12.4 %
C	non-coqdoc comments	5,661	300,850	5.2 %
E	coqdoc comments	2,910	137,401	2.7 %
S	imports & sectioning	2,073	49,377	1.9 %
L	context	1,329	50,686	1.2 %
D	definitions	8,778	308,858	8.0 %
N	notation	1,047	40,293	1.0 %
H	automation: directives	1,157	45,680	1.1 %
X	automation: program code	2,648	94,556	2.4 %
T	theorem statements	11,781	541,836	10.8 %
P	proof lines	58,157	1,981,655	53.3 %
	total	109,072	3,573,648	100.0 %

3.4 Mizar

The statistics in this paper are on version 7.8.05 of the Mizar system, which is distributed with version 4.87.985 of the MML mathematical library.

Mizar files have the suffix '`.miz`'. (There also are files with suffix '`.abs`' that are 'abstracts' to the formalizations, but they are derived from the first kind of files and do not contain any independent information.) As the version number of the MML library already shows, there are 985 `.miz` files distributed with the system.

Therefore the number of files counted were:

985 files `mml/*.miz`

And the statistics about these files were:

		lines	*bytes*	
B	blank lines	84,609	87,744	4.3 %
C	comments	10,857	488,821	0.6 %
S	environments, cancellations	23,655	1,118,819	1.2 %
L	reservations	5,396	194,693	0.3 %
D	definitions	56,738	1,847,161	2.9 %
N	notation	1,634	45,598	0.1 %
H	registrations	26,016	749,427	1.3 %
T	theorem statements	177,829	6,625,141	9.0 %
P	proof lines	1,582,831	61,096,949	80.4 %
	total	1,969,575	72,254,353	100.0 %

4 Categories of lines in a formalization

In the previous section we tagged lines in different systems that had similar functions with the same letter. Here we identify how these letters should be interpreted.

For most of the categories we list the main keywords that are associated with the lines of that category. For a user of the system this makes it quite clear how we divided the lines among the categories. (In Mizar there was the clearest bijection between keywords starting a part of a formalization and categories in our statistics. In the other systems the correspondence was a bit less obvious.)

4.1 Non-content lines

B – blank lines. Lines tagged 'B' are blank lines. These amount to a surprising large part of the total line count of a formalization. In the byte counts of this category we also included the white space at the end of other kinds of lines. Also sometimes some care had to be taken with files that did not end in a newline character. (For such files one newline byte was added.)

C – comments. The following table shows the comment styles found in the four systems:

HOL Light:	`(*` *comment* `*)`
Isabelle/HOL:	`(*` *comment* `*)`
Coq:	`(*` *comment* `*)`
Mizar:	`::` *comment*

E – documentation Isabelle and Coq generate documentation for the formalizations by having text inside special comments. In Isabelle these comments come in two styles, and are always prefixed with a keyword or a double dash. At first we included some of these lines in the sectioning category below, but Makarius Wenzel convinced us in private communication that they really belong in this category.

Isabelle/HOL:	`header section subsection subsubsection text txt --`
	`{*` *text* `*}` `"`*text*`"`
Coq:	`(**` *text* `*)`

4.2 Modules

Imports, sectioning and modules seem closely related, but there is a gray area with the notion of definitions. For instance in Isabelle a 'locale' might be considered to group related definitions together, but it also might be considered to consist of definitions. (We chose the second interpretation.) Similarly Coq modules seem rather close to Coq structures. (We chose to consider the first to be about modularization and the second to be a data-type definition.)

S – imports and sectioning. We did not distinguish between lines that group parts of a formalization together into a section or module, and lines that open or import these sections or modules.

The main keywords for this category of lines in the four systems were:

HOL Light: `needs loadt`
Isabelle/HOL: `theory imports begin use uses`
Coq: `Require Section Module Import`
Mizar: `environ begin canceled`

One could also consider Isabelle's `use` and `uses` to belong to the automation category below, but we decided to consider them to be import lines.

4.3 Definitions

L – contexts. The 'L' lines build 'contexts' in which a definition can be made. We considered these lines to be part of those definitions. In the Isabelle system these contexts are named entities. In Coq they just are implicit through the position in the section or module. In Mizar we used this letter for lines that introduce variable conventions.

Isabelle/HOL: `class locale context`
`instance interpret interpretation`
Coq: `Variable Variables Hypothesis Parameter Axiom`
Mizar: `reserve`

D – definitions. The systems all have numerous constructions for defining functions, predicates and types. Here are the main keywords for these constructions:

HOL Light: `new_definition new_recursive_definition define`
`new_inductive_definition new_specification`
`new_type_definition`
Isabelle/HOL: `abbreviation axclass coinductive constdefs consts`
`datatype definition defs fun function inductive`
`inductive_set nominal_datatype nominal_inductive`
`nominal_primrec primrec recdef record specification`
`typedecl typedef types`
Coq: `Definition Fixpoint Inductive CoFixpoint CoInductive`
`Record Function`
Mizar: `definition`

N – notation. These are the lines that direct the parser and pretty-printer of the system. These lines do not define the notions themselves, but introduce the syntax for the defined notions.

HOL Light: `parse_as_infix unparse_as_infix parse_as_binder make_overloadable overload_interface override_interface reduce_interface prioritize_num prioritize_real`
Isabelle/HOL: `syntax translations notation nonterminals parse_translation print_translation`
Coq: `Infix Notation` 'Reserved Notation' 'Tactic Notation' `Coercion` 'Implicit Arguments' 'Set Implicit Arguments' 'Unset Implicit Arguments' 'Set Strict Implicit' 'Unset Strict Implicit' 'Open Scope' 'Open Local Scope'
Mizar: `notation`

4.4 Automation

The automation of a system has two kinds of lines. First there are the lines that set parameters for the automated decision procedures and proof search procedures. Second there are the implementations of these automated procedures.

Most of the automation is implemented outside of the formalizations and is not counted here, but procedures that are specific to the subject are often implemented inside the formalization.

H – automation: directives. The automation 'directives' often are mixed with statements. For instance, in Isabelle theorem statements can be annotated with '`[simp]`'. This really is an automation directive, but it does not have a line of its own, so it will not be reflected in the statistics for this category of lines. Similarly, the Mizar 'registrations' (which direct the automation of the Mizar type system) also can be read as statements. For this reason in the Mizar pie chart on page 143 this category was included in the group about statements, and not in a group about automation.

Isabelle/HOL: `declare lemmas theorems`
Coq: `Hint Add Opaque Transparent Scheme`
Mizar: `registration`

X – automation: program code. These lines are implementations of proof procedures. In the HOL Light system really *all* lines are in some sense in this category, as a HOL Light formalization really just is an OCaml program. Therefore in the case of HOL Light the lines of this category are what remains when the other categories are removed.

The Mizar system does not have this kind of line as Mizar does not support user level proof automation.

HOL Light: `let`
Isabelle/HOL: `ML ML_setup declaration method_setup oracle setup simproc_setup`
Coq: `Ltac`

4.5 Theorems

A formalization mainly consists of a long chain of 'lemmas'. These lemmas generally consist of a label, a statement and a proof.

T – theorem statements. In this category are the lines which state the theorem and give its label.

The Mizar system actually distinguishes between two kinds of statements: theorems and schemes. The first category are the first order statements, while the second category are the higher order statements. Here we do not distinguish between these two categories.

HOL Light: `prove prove_by_refinement`
Isabelle/HOL: `lemma theorem inductive_cases axioms axiomatization corollary subclass termination`
Coq: `Lemma Theorem Goal`
Mizar: `theorem scheme`

P – proofs. Finally there are the lines of the formalized proofs. As is apparent in the pie charts in Section 2 these lines amount to about half of the formalizations. These lines contain many different constructions all with their own keywords. Here we just give the keywords that bracket the proofs.

Isabelle/HOL: `apply by proof qed done oops sorry`
Coq: `Proof Qed Save Defined`
Mizar: `proof end`

5 Conclusions

5.1 Discussion

The three main conclusions of this study for us are:

- The four systems are quite similar. Despite a large difference in foundations (the HOL logic, the Calculus of Inductive Constructions and Tarski-Grothendieck set theory are all quite different) and in interaction style (talking to an OCaml interpreter, interacting with a tactic prover in a Proof General style interface and using a compiler-like batch checker), the actual formalizations all share the same elements.
- The HOL Light system has the smallest definition segment in its pie chart. This seems to suggest that it is the most reliable. Andrzej Trybulec taught me in private communication that a definition is like a *debt*, because you do not know whether what you are defining corresponds to the informal notion in your head. You gain confidence in this by proving theorems about the notion later. That way you pay the debt back, and gain trust in that your formalization actually means what you think it means. In this sense the HOL Light system is the most trustable of the four.

Another interpretation, proposed by John Harrison in a private communication, is that the low percentage of the definitions does not so much reflect the quality of the formalization but rather the fact that the HOL Light library primarily contains pure mathematics. This seems to be collaborated by the observation that the percentage of the definitions in John Harrison's verification work at Intel, also using the HOL Light system, is 3.7% instead of 1.3%.
- The Mizar system has the largest proof segment in its pie chart. This suggests that its proof language might be less efficient. (It is very natural and pleasant to use, though.) This might be related to Mizar's declarative proof style, or the fact that the Mizar system does not have much automation.

5.2 Future work

It might be interesting to delve into the 'fine structure' of the largest segment in the pie charts, the proof lines. However, it probably is hard to systematically distinguish different kinds of proof steps on a line by line basis. An interesting question about the proof lines might be how many are straight-forward 'manual' reasoning steps, and how many invoke strong automated proof procedures.

Acknowledgments. Thanks to John Harrison, Henk Barendregt and Makarius Wenzel for helpful comments. Special thanks to John Harrison for sending me statistics on his Intel verification work. Special thanks to Makarius Wenzel for sending me a list of categorized Isabelle keywords.

References

1. Harrison, J.R.: *The HOL Light manual (1.1)*, 2000. <http://www.cl.cam.ac.uk/users/jrh/hol-light/manual-1.1.ps.gz>.
2. Muzalewski, M.: *An Outline of PC Mizar*. Fondation Philippe le Hodey, Brussels, 1993. <http://www.cs.ru.nl/~freek/mizar/mizarmanual.ps.gz>.
3. Nipkow, T., Paulson, L.C., and Wenzel, M.: *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002. <http://www.cl.cam.ac.uk/Research/HVG/Isabelle/dist/Isabelle2004/doc/tutorial.pdf>.
4. The Coq Development Team. *The Coq Proof Assistant Reference Manual*, 2006. <http://pauillac.inria.fr/coq/doc/main.html>.
5. Wiedijk, F., editor: *The Seventeen Provers of the World*, volume 3600 of *LNCS*. Springer, 2006. With a foreword by Dana S. Scott.

Author Index