From editor

This issue presents proceedings of the workshop "30 Years of Mizar"[1] held in September 18, 2004 in Białowieża (Poland). The workshop was devoted to celebrate 30 years of Mizar. It was affiliated to the Third International Conference on Mathematical Knowledge Management. The workshop was organized by University of Białystok and Białystok Technical University.

The papers were selected for presentation at workshop (with two exemptions) and publication in the journal by workshop's program committee which consists of the following:

- Grzegorz Bancerek (Chair), Bialystok Technical University, Poland,
- Czesław Byliński, University of Bialystok, Poland,
- Adam Grabowski, University of Bialystok, Poland,
- Roman Matuszewski, University of Bialystok, Poland,
- Yatsuka Nakamura, Shinshu University, Nagano, Japan,
- Piotr Rudnicki, University of Alberta, Edmonton, Canada,
- Yasunari Shidama, Shinshu University, Nagano, Japan,
- Christoph Schwarzweller, University of Gdansk, Poland,
- Freek Wiedijk, Radboud University Nijmegen, The Netherlands.

The first exception concerns the paper of Paul Cairns which was submitted to the parallel workshop "Mathematical User-Interfaces" and was presented at shared session of both workshops as being of common interest. The second exception concerns the invited talk by Yasunari Shidama.

At the workshop there were also presented three talks which are not published in this issue:

- *Mizar in Secondary Schools* by Anna Rybak,
- *Towards a Mizar Mathematical Library in OMDoc Format* by Michael Kohlhase,
- *Mizar project wishlist* by Josef Urban.[2]

Grzegorz Bancerek

[1] Workshop's web page may be found at **http://merak.pb.bialystok.pl/mkm2004/**

[2] The wishlist may be found at **http://wiki.mizar.org/cgi-bin/twiki/view/Mizar/MizarWishlist**

MIZAR: the first 30 years

Roman Matuszewski[1] and Piotr Rudnicki[2]*

[1] University of Białystok, Białystok, Poland, **romat@mizar.org**
[2] Dept. of Computing Science, University of Alberta, Edmonton, Canada, **piotr@cs.ualberta.ca**

The MIZAR project is *opus magnum* of Andrzej Trybulec.

Abstract – We present the story of the MIZAR project with focus on the years until 1989. A lot about MIZAR after 1989 is available at the web[3].

1. Introduction

In 1967 Andrzej[4] started working at the Płock Branch of Warsaw University of Technology in Płock (100 km North-West of Warsaw). It is here that we met Andrzej for the first time when we entered the university: the first author in 1969 and the second in 1968. Andrzej was our math teacher of calculus for engineers.

We remember Andrzej from these early days from the Informatics Club (in Polish: Koło Naukowe ETO) that he ran for several years. The Club met quite frequently to discuss widely understood issues of informatics. In those years, it seemed like everyone wanted to design their own programming environment and needless to say Andrzej was planning to have his own, too. This project was short lived yet something remains of it: its name—MIZAR— which, to the best of our memory, appeared in late 1972. According to Andrzej, it was his wife Zinaida who picked the name. She was looking through an astronomical atlas when Andrzej asked her for a good name for a project and she suggested MIZAR, the name of a star in the familiar Big Bear constellation.

2. 1973-74: the very beginnings

Andrzej was finishing his PhD in topology at the time and apparently the final stages of this effort provided a strong motivation for his envisaging a computerized assistance in the process of editing mathematical papers. During September-October of 1973, Andrzej was visiting Institute of Scientific and Technical Information (VINITI) in Moscow where he discussed his ideas. The first presentation of the MIZAR ideology— ideology understood here as visionary speculation—was presented by Andrzej on November 14, 1973 at a seminar in the Institute of Library Science and Scientific Information at Warsaw University. During the seminar Andrzej postulated a language for recording mathematical papers such that:

- the papers could be stored in a computer and later, at least partially, translated into natural languages,
- the papers would be formal and concise,

* Partially supported by NSERC grant OGP9207.

[3] **http://mizar.org**

[4] When we write Andrzej, we mean Dr. Trybulec.

Manuscript received August 5, 2004; revised September 9, 2004

- it would form a basis for the construction of an automated information system for mathematics,
- it would facilitate detection of errors, verification of references, elimination of repeated theorems, etc.
- it would open a way to machine assisted education of the art of proving theorems,
- it would enable automated generation of input into typesetting systems.

It is worth stressing that in this initial stage, the question of proof-checking has barely been mentioned; the main stress was placed on editorial work. Andrzej initiated a group effort of translating, into the yet non-existing language, a paper by H. Patkowska, *A homotopy extension theorem for fundamental sequences*, Fund. Math. 64 (1969), pp. 87-89. In the long term, this strategy proved to be the best way to arrive at a practical language for formalizing mathematics.

The first experiments with implementing a version of MIZAR for propositional logic started in the Fall of 1974 by Andrzej, Krzysztof Łebkowski and Roman Matuszewski on the Polish made machine ODRA-1204 in Algol 1204. Since the grammar of MIZAR was quite unstable, a universal syntax analyzer was implemented, with a rather atrocious running time of generated parsers.

3. 1975: MIZAR-PC

The above mentioned experiments continued through 1975. In June 1975, we obtained some financing from Płock Scientific Society.[5]

In June 1975, Andrzej circulated a short write-up entitled *Logic-information language* MIZAR. It was the first written document with clearly stated MIZAR ideology—see above—and some details of the implementation which was in progress at that time. Andrzej presented his ideas at IX. Kolloquium über Information und Dokumentation, November 12–14, 1975, in Ilmenau, East Germany, which resulted in the first MIZAR publication [9]. At that time even these ideas were surrounded by an aura of science-fiction, although it was soon discovered that these ideas were by no means new, one can find them in a short 1962 paper by Kaluzhnin [2], which was brought to Andrzej's attention around 1975–76.

A preliminary version of a report on MIZAR-PC (PC for propositional calculus) was presented to the Society in November 1975. The report included a description of a language for recording proofs in classical propositional calculus in the Jaśkowski style of natural deduction. Actually, it was only a few years later that we learned about the Jaśkowski style of natural deduction, however it was the way the proofs had been written in the tradition of the Polish mathematical school. Here is a text from the 1975 report:

```
begin
((p ⊃ q) ∧ (q ⊃ r)) ⊃ (p ⊃ r)
proof
    let   A: (p ⊃ q) ∧ (q ⊃ r) ;
    then  B: p ⊃ q ;
          C: q ⊃ r by A ;
    let      p ;
    then     q by B ;
    hence    r by C
end
end
```

[5] In Polish: Towarzystwo Naukowe Płockie. This society is one of the oldest regional societies of this type in Poland, established in 1820, see **http://www.tnp.plocman.pl**.

Please note that this is a true picture of the texts written in MIZAR-PC: the teletype used for the input to the machine we used provided all the characters displayed above and also made underlining possible, which was the tradition for writing down keywords in implementation of Algol that we used.

MIZAR-PC texts started with **begin** and were terminated by **end**, although these two keywords also played other roles. Besides checking syntax the implemented analyzer of MIZAR-PC was also checking correctness of proofs (**proof** ... **end**) and inference steps (... **by** ...).

Checking the correctness of proofs consisted of checking the equality of the sentence to be proved and what was *the contents of a proof*. The contents of a proof is a sentence extracted from assumptions (**let** ...) and conclusions (**thus** ... or **hence** ...) occurring in the proof. A proof could have a number of assumptions and conclusions. In constructing the contents of a proof, an implication was placed after an assumption and a conjunction after a conclusion (except the last one).

MIZAR-PC offered a construct called a *compound statement* bracketed by symbols **begin** ... **end** between which all constructs allowed in a proof could be placed. The contents of a compound statement consisted of a sentence constructed in the same way as in the contents of a proof. A compound statement could be labeled and a reference to the label meant a reference to its contents. Loosely speaking, a compound statement was like a proof without explicitly stating what was being proved.

The process of checking of inference steps was based on a fixed set of rules of inference (some five hundred of them). An inference rule was a scheme of acceptable inference and could have up to two premises, each with at most one binary connective and a conclusion, also with at most one binary connective. No more than three propositional variables were permitted in an inference rule. An accepted inference step had to be an instance of exactly one of the allowed rules of inference. This approach was abandoned in the next MIZAR version.

MIZAR-PC introduced *linkage*, a mechanism for making reference to the *previous* sentence without using a label. The keyword **then** served this end. **hence**, one of the keywords marking a conclusion, has meaning equivalent to **thus** **then** which was not permitted.

It is worth mentioning that MIZAR-PC foresaw references to a data base, although this feature was not implemented until 1989.

The implementation of MIZAR-PC was on a Polish computer ODRA 1204 with 12k of 24 bit words and a drum of 192k such words, in Algol 1204. The input medium was paper tape or a teletype. The members of the team implementing MIZAR-PC: Roman Matuszewski, Piotr Rudnicki and Andrzej Trybulec.

During 1975–76, MIZAR-PC was used in teaching propositional logic at the Płock Branch of Warsaw University of Technology (Roman Matuszewski) and at the Institute of Library Science and Scientific Information at Warsaw University (Andrzej Trybulec).

4. 1977: MIZAR-QC/1204 and MIZAR-QC/6000

The next natural step in developing MIZAR was to furnish the language with quantifiers. The work continued over 1976 when Andrzej moved from Płock to Białystok and started working at the Białystok Branch of Warsaw University, where he has been working until now. Since 1997 this school has been known as University of Białystok.

Despite lack of financing, the work continued on extending MIZAR-PC towards quantifier calculus. Under Andrzej's supervision some longer texts were written and used in guiding the development of the system:

1. Jan Borawski, in his MSc thesis, June 1977, formalized a paper by J. Krasinkiewicz *On homeomorphism of the Sierpiński curve*, Commentationes Mathematice XII, 1969, pp. 255–257.
2. Chinese remainder theorem from [1], by Cz. Żukiewicz.

In 1977, Andrzej secured some financing for MIZAR through a research grant of the Ministry of Science and Higher Education, administered in our case by the Institute of Computer Science[6], Polish Academy of Sciences.

In late August 1977, the implementation of MIZAR-QC was completed in Pascal on a CDC 6000 by: Andrzej Jankowski[7], Roman Matuszewski, Piotr Rudnicki, Andrzej Trybulec.

The language of MIZAR-PC was extended with quantifiers to form MIZAR-QC. However, the language was still quite simplistic, not much more than quantifiers and their processing in proofs was added; here is a sample text

```
BEGIN
((EX X ST (FOR Y HOLDS P2[X,Y])) > (FOR X HOLDS (EX Y ST P2[Y,X])))
  PROOF
    ASSUME THAT A: (EX X ST (FOR Y HOLDS P2[X,Y]));
    LET X BE ANYTHING;
    CONSIDER Z SUCH THAT C: (FOR Y HOLDS P2[Z,Y]) BY A;
    SET Y = Z;
    THUS D: P2[Y,X] BY C;
  END
END
```

The universal and existential quantifiers were written as

FOR <*variable list*> **HOLDS** <*sentence*>
EX <*variable list*> **ST** <*sentence*>

respectively[8].

A fixed set of predicates was chosen for testing the language processor

CONTRADICTION, **P**, **Q**, **R**, **P1**, **Q1**, **R1**, **P2**, **Q2**, **R2**, **P3**, **Q3**, **R3**,

where the first four were nullary, while the arity of the remaining ones was indicated by the digit following the letter.

Variables and constants were assumed to denote objects from a fixed non-empty set. The only terms allowed were simple terms built of a variable or a constant.

Three syntactic constructs provided the means for introducing quantifiers when computing the *contents of a proof*. These constructs played a double role: 1) they affected the computation of the contents of a proof and 2) they also introduced objects used in the rest of the proof.

– The <*let statement*>

LET <*variable list*> **BE** <*specification*>

[6] Within the research program MR.I.3 we worked on the topic 02.5.1 "Logic-information language MIZAR-QC". This source of funding was supporting MIZAR until 1983.

[7] A logician who reduced his role to convincing us that MIZAR cannot be built.

[8] While everybody agrees that **EX** stood for *there exists*, there is no consensus whether **ST** comes from *such that* or *satisfying*.

introduced fixed but arbitrary objects to be used in the rest of the proof. The only permitted specification was **ANYTHING** and had to be stated as Andrzej did not like the look of just **LET X;** (which started to occur frequently in future MIZARs once the reservation of variables was introduced).

This statement contributed a universal quantifier to the contents of a proof with bound variables from the <*variable list*> and its scope was computed from the remaining proof elements.

– The <*consider statement*> had two syntactic variants

CONSIDER <*variable list*> or
CONSIDER <*variable list*> **SUCH** <*conditions*> <*justification*>

This was the MIZAR construct for recording existential elimination, i.e. the way of using existentially quantified sentences. Since the variables denoted objects from a non-empty set, the first variant of the statement was safe.

As a proof element, this statement introduced existential quantifiers into the contents of a proof, and thus played the role existential introduction.

– The <*set statement*>

SET <*variable list*> **=** <*argument*>

introduced a named object (or several of them) and equated them to its argument. This statement also played the role of existential introduction in computing contents of proofs.

In MIZAR-PC, **let** was used to indicate an assumption. Since **LET** was now used in a new role, the new keyword **ASSUME** was introduced to indicate an assumption. It was possible to assume several labeled sentences joined by **AND**.

The justification procedure, previously called the inference checker, had been completely redesigned. A justification had a general shape of

$$\beta \ \mathbf{BY}\ d_1, \ldots, d_n$$

where the designator d_i identified the label of a sentence α_i. The justification was perceived as correct if the following sentence

$$(\alpha_1 \supset \ldots (\alpha_n \supset \beta) \ldots)$$

was accepted by a justification procedure. The justification procedure was based on a set of rewrite-like rules and was implemented as such. Before any rewrite rules were applied, all sentences were transformed into a standard form with negation, conjunction and universal quantifier as the only connectives. In the standard form there were no double negations, no fictitious quantifiers and all quantifiers bound only one variable.

Because the running time of the justification procedure even for justifications not involving quantifiers could have been exponential in the number of propositional variables, a complexity value was assigned to each rewrite rule and a running sum of these values was kept during each run of the procedure. Whenever the sum exceeded certain (quite arbitrarily chosen) value, the justification procedure terminated, announcing that the task was too complicated and the examined justification was not accepted. The running time of the rules manipulating the substitutions for universally bound variables was particularly bad and could lead to the running time of the order of n^n, where n was the number of leading universal quantifiers, as all possible substitutions were blindly considered.

Whether a proof was acceptable was determined by running the justification procedure on the formula $\phi \supset \psi$ where ϕ was the contents of a proof and ψ was the sentence being proved, both sentences in the standard form.

The work was continued in 1978 and a number of simple formalizations had been carried out: set theory (simple facts about containment, union and intersection), lattice theory (simple facts about linear and partial orders phrased in terms of a lattice of sets), an attempt to translate several pages in foundations of geometry (later continued in MIZAR-MS).

5. 1978: MIZAR-MS

Even the limited experience in trying to use MIZAR-QC for recording mathematics prompted quite a number of changes as the language was too frugal for comfort. This led to MIZAR-MS[9] after a number of superficial syntactic extensions and a few more substantial additions.

The superficial syntactic extensions and changes included:

1. The propositional connectives written as: **NOT**, **&**, **OR** **IMPLIES** and **IFF**.
2. Sentence labels became optional.
3. Relaxed usage of parentheses (MIZAR-QC required almost full parenthesizing and no surplus parentheses were permitted).
   (a) Quantifiers were treated as connectives with lowest priority.
   (b) Parentheses were not required after negation.
   (c) In a compound formula with conjunctions (or disjunctions) as the only kind of connective, no parentheses were required and such connectives were right-associative.
4. Diffuse (compound) statements present in MIZAR-PC were reintroduced with the opening bracket **NOW**.
5. Global constants were allowed (in MIZAR-QC, **CONSIDER** was permitted only in proofs).
6. In assumptions, the use of **THAT** after **ASSUME** was not required for a single proposition as an assumption.
7. In universally quantified formulae, **HOLDS** can be omitted if the scope is an existential formula (**EX** ...).

There was also a number of substantial additions and changes.

1. Predicate definitions, syntactically

   FOR $\{<$*variables*$>$ **BEING** $<$*specification*$>\}^+$
   ST $<$*sentence*$>$
   PRED $<$*pred id*$>$ **DENOTES** $<$*definiens: sentence*$>$

   The arguments of the defined predicate were given by $<$*variables*$>$ typed by $<$*specification*$>$ in the listed order. It was also possible to introduce predicates with no arguments.
2. Scheme definitions, syntactically

   SCHEME $<$*scheme id*$>$ **;**
   PREDICATE $<$*pred id list*$>$ **;**
   CONSTANT $<$*const id list*$>$ **;**
   $<$*scheme claim*$>$
   SINCE
   $<$*formal premise*$_1$*: labeled sentence*$>$ **;**
   ...
   $<$*formal premise*$_k$*: labeled sentence*$>$ **;**
   PROOF ... **END**

[9] This effort was also financed by Płock Scientific Society.

A scheme is a pattern of theorems expressed in terms of formal predicates and constants. A specific theorem matching <*scheme claim*> is obtained after providing actual premises that appropriately match the formal ones given by <*formal premise*>s. In a justification, a scheme was used as follows

<*sentence*> SCHEME <*scheme id*> **(** *label*$_1$, ..., *label*$_k$ **)**

The <*sentence*> was accepted if it matched the <*scheme claim*> of <*scheme id*> and the sentences labeled *label*$_1$, ..., *label*$_k$ matched the premises of the scheme. The actual predicates and constants were automatically reconstructed from actual premises and the theorem being justified. There was no other means to specify the actual predicates and constants. Schemes were not fully implemented at that time.

3. Specification (type) declarations, only of the form

TYPE <*id*>

Every variable was typed either by the predefined specification **ANYTHING** or a declared type. This feature was responsible for MS in MIZAR-MS, namely Multi Sorted.

4. The keyword **TAKE** replaced **SET** (from MIZAR-QC) and became the only statement introducing existential quantifiers when computing the contents of a proof. The role of the **CONSIDER** statement was reduced to existential elimination only.
5. Absolute equality **=** and inequality **<>** were predefined. While equality was automatically processed as an equivalence relation, inequality was not processed as a symmetric relation. (As a curiosity we would like to mention that in **BY** justifications, labels designating equalities had to be listed last.)

Two larger texts were developed:

1. Elżbieta Ramm and Edmund Woronowicz proved the correctness of a factorial computing program using the Winkowski method of reasoning about programs, their work appeared later as [5].
2. Jerzy Zabilski and Roman Matuszewski translated pages 27–38 of *Foundations of geometry* by K. Borsuk and W. Szmielew.

One of the problems faced in all of these formalizations was caused by the lack of any support for stating the axioms of the theory one wanted to work in. Two workarounds were used: either stating the entire development within one compound statement with axioms as assumptions or stating the axioms at the main text level (typically at the very beginning) and letting the analyzer report that they were not accepted. This problem was partially resolved in MIZAR-2 (1981) and finally in PC-MIZAR (1988–89).

At that time, the MIZAR group started to grow substantially while anchored at the Białystok Branch of Warsaw University. MIZAR-MS was implemented by Czesław Byliński, Piotr Rudnicki, Andrzej Trybulec, Edmund Woronowicz and Stanisław Żukowski on a CDC 6000 in Pascal/6000.

6. 1978–79: MIZAR-FC

Until 1978 MIZAR had been lacking functional notation and therefore had only simple terms. The situation has changed now.

6.1 Function definitions

Two syntactic constructions served to introduce functions:

1. <*choice statement*> introduced the so called choice functions and had the following syntax

   <***FOR**-prefix*>
   CONSIDER <*choice list*>
   SUCH <*conditions*>

   The names of defined functions were given by identifiers from <*choice list*> and their arguments were given by (optional) <***FOR**-prefix*>. For example:

```
FOR X, Y CONSIDER F, G SUCH THAT Z1: F <> G AND
                                 Z2: B[X,F,Y] AND
                                 Z3: B[X,G,Y];
```

   introduced binary functions **F** and **G** which were used to build terms, e.g., **F(A,B)**.
   A <*choice statement*> with <*conditions*> had to be justified and for the above we had to justify the existence of **F** and **G**, i.e., we had to prove that:

```
FOR X, Y EX F, G ST F <> G & B[X,F,Y] & B[X,G,Y]
```

   In further text, a reference to one of the labels in <*conditions*> above, say **ăZ3**, referred to the following sentence:

```
FOR X, Y HOLDS B[X,G(X,Y),Y]
```

   Here is a more tangible example of defining the intersection of two sets:

```
FOR X, Y BEING SET
  CONSIDER CAP BEING SET SUCH THAT
    CAPCOND: FOR Z BEING INDIVIDUAL
               HOLDS IN[Z,CAP] IFF (IN[Z,X] & IN[Z,Y]);
```

2. <***TAKE** statement*> played a double role and had the following syntax

   <***FOR** prefix*>
   TAKE <***TAKE** list*> **=** <*expression*>

   The <***FOR** prefix*> was optional. Without this prefix, the <***TAKE** statement*> played a role analogous to the <***TAKE** statement*> introduced in MIZAR-MS.
   The <***TAKE** statement*> with the <***FOR** prefix*> introduced functions or operations whose behavior was defined by an expression. One can think of this statement as corresponding to a λ definition. For example:

```
FOR X BEING INTEGER TAKE SUCC = X+1;
```

   introduced one unary function to be used for terms like **SUCC(A)**. The type returned by the function was inferred from the type of the expression; the arguments were defined by the <***FOR** prefix*>.
   The binary **+** for **Integer**s was predefined. The functions defined in this fashion were automatically expanded to their definiens when necessary and thus the following statement did not require additional justification

```
SUCC(SUCC(X+Y)) = ((X+Y)+1)+1
```

The <***TAKE*** *statement*> was also used to define operations as an alternative notation for functions. For example:

```
FOR X, Y BEING REAL CONSIDER MULT BEING ELEMENT
. . .
FOR X, Y BEING REAL TAKE X*Y = MULT(X,Y)
```

The set of allowed operation symbols, unary and binary, was predefined and was quite small.

6.2 Relation definitions

Similar to the method of defining operations, predicates could be defined using relational symbols. The set of allowed binary relational symbols was fixed and quite small. For example, set inclusion was defined as

```
FOR X, Y BEING SET PRED X <= Y DENOTES
   FOR E BEING INDIVIDUAL HOLDS IN[E,X] IMPLIES IN[E,Y]
```

and the less than or equal for integers as

```
FOR X, Y BEING INTEGER PRED X <= Y DENOTES
   EX Z BEING INTEGER ST Z > 0 & X+Z = Y+1
```

where relation **>** had to be defined earlier.

While the above definitions use the same relational symbols, they define two different relations as the types of arguments differ.

6.3 Other changes

– The predeclaration feature allowed text to be shortened as one did not have to specify the type of a variable at its defining point if the name of the variable was predeclared earlier to be of some type. For example, with the predeclaration

```
LET X, Y, Z DENOTE SET;
```

the sentence **FOR X, Y, X HOLDS ...** was equivalent to the sentence **FOR X, Y, Z BEING SET HOLDS ...**

Predeclarations were a precursor of the current reservations.

– The schemes proposed in MIZAR-MS were not implemented because an attempt to incorporate functions into schemes forced a change in the very idea of how to implement them.

– Type **INTEGER** was predeclared as well as binary operations **+**, *****, **−** and binary relations **<**, **<**, **<=** and **>=**. However, their properties had to be given explicitly in each MIZAR-FC text.

– Some syntactic means were introduced to exclude a part of the text from proof-checking. The text between the following pragmatic comments

```
(*$J−*)
   . . .
(*$J+*)
```

was checked only for syntactic correctness. It was intended to be a place for declaring functions and relations and stating axioms defining these notions.

This feature was also used to shorten processing time of longer texts by temporarily switching off checking for parts of the text which were already finished.

– The rewrite-rules based justification procedure of MIZAR-QC was abandoned in favor of model checking, albeit quite naively implemented. In a justification problem we are to decide whether or not $\beta \supset \alpha$ is a tautology (a negative answer did not always mean that that was not the case). The sentence was converted into $\beta \wedge \neg\alpha$ and the procedure looked for a contradiction. The sentence was first converted into a standard form (like in MIZAR-QC). Then all atomic and universal sentences (collectively called basic sentences) were collected, at this stage the scopes of universal quantifiers were not inspected. There was a limit ($n \leq 10$) on the number of such sentences. In the next step all 2^n cases of possible logical valuations of these sentences were considered and each was checked to see whether or not it led to a contradiction. In this last stage, possible instantiations of positively occurring universal sentences were considered, one such sentence at a time, which meant that the universal sentences did not "cooperate" in the process.

MIZAR-FC was used to record a number of larger texts. Among these texts was the initial segment of the book on arithmetics by Grzegorczyk [1]. The book was so rigorous and detailed that the blow-up factor in the translation to MIZAR-FC was reasonably small. This effort lasted for several months with participation of Andrzej Trybulec, Czesław Byliński and Stanisław Żukowski.

Elżbieta Ramm and Edmund Woronowicz, [5] rewrote their earlier developments in MIZAR-MS into MIZAR-FC on building an environment for proving properties of programs.

MIZAR-FC was implemented in Pascal/6000 by Czesław Byliński, Roman Matuszewski, Elżbieta Ramm, Piotr Rudnicki, Andrzej Trybulec, Edmund Woronowicz, Stanisław Żukowski.

7. 1981: MIZAR-2

The experience of all previous MIZARs resulted in Andrzej's design of a language simply called MIZAR whose processor was team implemented on ODRA 1305 (ICL 1900) with contributions by Czesław Byliński, Henryk Oryszczyszyn, and Piotr Rudnicki. The first release of the system on July 10, 1981 was quickly followed (nobody seems to be sure why) by the second release on September 28, 1981. This release was further called MIZAR-2 and in the following years was ported to quite a number of different machines, e.g., mainframe IBM and UNIX. MIZAR-2 offered substantial improvements over its predecessors and also has had quite a future: its reimplementation in 1986 as MIZAR-4 directly led to the current MIZAR.

A MIZAR-2 text was split into two sections

environ
    MIZAR statements with no justifications,
    syntactic checking only
begin
    statements with justifications

Note however, that each MIZAR-2 text was a stand-alone unit and there was no possibility of information flow between articles (besides copying text). The environment section was the place to state all the machinery and facts needed for developments in the text proper. Since the environment section was checked only for syntactic correctness, it was not unheard of that someone stated a false claim making further proving more convenient.

We resort to several illustrative examples in presenting more important features of MIZAR-2 as a more general description would require substantial space.

7.1 Types

Several syntactic means were available for defining new types of objects. The simplest of them was just introducing a name for a longer type expression, e.g.,

```
TYPE RELATION DENOTES SUBSET OF [U,U];
```

In a more complicated definition one could specify a type as a set of all objects satisfying certain conditions

```
TYPE MAP OF A,B BEING NONEMPTY
  INCLUDES F BEING SUBSET OF [A,B]
    SUCH THAT
    AXF1: FOR X BEING ELEMENT OF A
             EX Y BEING ELEMENT OF B ST [X,Y] IN F          AND
    AXF2: FOR X BEING (ELEMENT OF A), Y1, Y2 BEING ELEMENT OF B
             ST [X,Y1] IN F & [X,Y2] IN F HOLDS Y1=Y2
PROOF ... END
```

One had to prove non-emptiness of such a type by demonstrating the existence of the sample object with the desired properties.

One could also define structures, e.g., the first step in defining a field

```
TYPE FIELDSHAPE CONSISTS OF
  UNIV BEING NONEMPTY,
  ADD, MULT BEING (RELATION OF PAIRS(UNIV),UNIV),
  O, E BEING (ELEMENT OF UNIV);
```

This was stated in the environment and the needed field properties were then given as axioms. Given a **FIELDSHAPE F** one could refer to its components as e.g. **ADD(F)**.

7.2 Definitions of functions with an explicit format

Some freedom was added into the way the format of a function is defined. Namely, in the case of choice functions, the arguments of a function were all variables from the **FOR** prefix in the listed order but this restriction is now removed.

```
DEFINITION LET A,B BE NONEMPTY, X BE (ELEMENT OF A),
              Y BE (ELEMENT OF B), F BE MAP OF A,B;
  PRED Y = VALUE(F,X) DENOTES [X,Y] IN F
    PROOF
      THUS EX Y BEING ELEMENT OF B ST [X,Y] IN F BY ... ;
      THUS FOR Y1,Y2 BEING ELEMENT OF B
             ST [X,Y1] IN F & [X,Y2] IN F HOLDS Y1 = Y2 BY ...
    END
END;
```

In the above definition the format of the function is given explicitly: the function has two explicit arguments, although it really has four arguments: **A**, **B**, **X** and **F**. A proof of existence and uniqueness was required for a defined function.

There was no means to make an explicit reference to the definiens of a function, however, the following sentence was obvious (for appropriate arguments)

```
[X,Y] IN G IFF Y = VALUE(G,X);
```

7.3 Definitions per cases

New syntax was added which helped to organize a definiens into a more readable phrase than just a long conjunction of implications, e.g.

```
DEFINITION
 LET S BE SIMPLANE, A, B, X, Y, Z BE ELEMENT OF POINTS(S)
         SUCH THAT Z: A<>B;
  PRED Z IS CONJ OF A, B, X DENOTES
    WHEN X=A => Z=A
    WHEN X=B => Z=B
    OTHERWISE [[A,X,Y],[A,B,Z]] IN NEGSIM(S)
END;
```

A definition given by cases required a justification of consistency. Although definitions per cases were meant to be available also for functions, they became fully implemented in PC-MIZAR of 1989.

7.4 Schemes

The schemes proposed in MIZAR-MS are now fully implemented and the scheme of induction is finally available

```
SCHEME INDUCTION;
    PRED P;
  FOR K BEING NATURAL HOLDS P[K]
    SINCE
    A: P[1];
    B: FOR K BEING NATURAL ST P[K] HOLDS P[SUCC(K)]
END;
```

This scheme was always assumed in the environment as proving it in MIZAR-2 would be quite a challenge. However, there were a number of defined existence schemes. E.g., the following scheme was useful in proving the correctness of the definition of a function

```
SCHEME RELDEF;
  PRED P;
  (EX R BEING RELATION
     ST FOR X,Y BEING A HOLDS [X,Y] IN R IFF P[X,Y])            &
  (FOR R,L ST ((FOR X,Y BEING A HOLDS [X,Y] IN R IFF P[X,Y])  &
               (FOR X,Y BEING A HOLDS [X,Y] IN L IFF P[X,Y]))
      HOLDS R=L)
  PROOF ...  END;
```

and its proof used the scheme of separation of subsets which was declared in the environment.

7.5 Justification procedure

The justification procedure was still a disprover looking for a model of a formula obtained after negating the sentence to be justified and conjuncting it with all the sentences used as premises. However, internally the procedure has undergone a substantial remake.

Firstly, the procedure did not blindly consider all possible valuations of basic sentences but rather tried to compute a valuation which would provide the sought for model. This process involved performing joins and intersections of lists of valuations.

Secondly, the procedure did not blindly consider all possible substitutions for bound variables. Instead, a sort of pattern matching procedure was performed to find "promising" substitutions by matching basic sentences with their counterparts in the scope of a quantifier and containing bound variables. The collected substitutions were then subjected to similar list manipulations of joins and intersections as the valuations above.

One restriction remained: in searching for a model the procedure never simultaneously considered substitutions into more than one universal sentence. This had the drawback of not using the full power of unification but it had the advantage of an inference checker running very fast. The latter was happening at the expense of the MIZAR author being forced to write small inference steps.

7.6 Miscellany

- Symbols of relational operators and operations were fixed and the language did not provide any means for adding new ones.
- The keyword **THEOREM** could have preceded a (labeled) sentence but did not play any role otherwise. There was some discussion to introduce other similar keywords like: proposition, lemma, corollary, etc. It has not happened until now.
- The **RECONSIDER** statement allowed for a change in the type of an object (this required a justification).
- Several notions, or rather notations only, were predefined
  - set theory: **CLASS**, **SET**, **IN NONEMPTY**, **ELEMENT OF** and **SUBSET OF**.
  - arithmetic: the sets **NATURALS**, **INTEGERS** were predefined as well as **NATURAL** being a shorthand for **ELEMENT OF NATURALS** and **INTEGER** for **ELEMENT OF INTEGERS**.
  - Small natural constants (up to 9999999).

  Only the most rudimentary (if any) properties of these notations were available automatically, all essential ones had to be stated in each text that used them.
- Operational brackets **[** and **]** for constructing expressions, intended to be used for n-tuples (e.g., pairs, Cartesian products) and type aggregates (e.g., fields).
- Attributive format for predicates such that one could write: **X IS COMPACT** or **U IS NEIGHBORHOOD OF W**.
- Iterative equality.

7.7 Formalizations

The translations into MIZAR-2 included

- *On the homotopy types of some decomposition spaces* by K. Borsuk, formalized by A. Trybulec. This development is continually being maintained and its final version is included in MML as [15].
- A proof that a field with conjugate and a plane with similarities are mutually interpretable by K. Prażmowski and P. Rudnicki.
- Pigeonhole principle, by P. Rudnicki.
- Basics of set theory and theory of relations, by Z. Trybulec.
- Basics of general topology, by Cz. Byliński.

In all of these formalizations, the stress was on proving theorems rather than developing types and auxiliary functions which were usually just stated in the environment without the burden of having to justify their correctness.

In the following years, MIZAR-2 was also applied to prove properties of programs [7] and software specifications [6]. The approach was based on natural, operational semantics of programs proposed by R. Burstall and J. Winkowski.

On December 13, 1981, martial law was declared in Poland. A side effect was our limited access to computers which turned out to be a blessing in the long run as finally there was some time to order a lot of thoughts and designs. For several months we were deprived (like all other people in Poland) of telephone connections and inter-city travel was troublesome as special permits were imposed. This had an adverse effect on the MIZAR team which was split between Białystok and Warsaw.

8. 1982: MIZAR-MSE

Numerous experiments with MIZAR-2 indicated that the language was satisfactory for recording some kinds of mathematics. Unfortunately, the semantics of the entire system seemed too complicated and nebulous for precise description. This prompted Andrzej to define a small sub-language of MIZAR. The sub-language included the well tested and frequently used constructs that were also amenable for complete and precise description [10].

The sub-language was named MIZAR-MSE and covered multi-sorted predicate calculus with equality, thus MSE. There was no functional notation in MIZAR-MSE, no definitions for predicates or schemes.

The first version of MIZAR-MSE was implemented by Roman Matuszewski, Piotr Rudnicki and Andrzej; numerous further and substantially different implementations followed, too many to mention here.

It was hoped that MIZAR-MSE could be used in teaching logic and some fragments of mathematics and indeed the hope was materialized as MIZAR-MSE was used at many universities all over Europe and North America. One of the frequently voiced criticisms of MIZAR-MSE was that it was too far removed from the mathematical practice and recently we are witnessing a general switch to using “full” MIZAR in teaching. Although quite a number of longer texts were written in MIZAR-MSE and distributed to users, these texts were stated in a very frugal notation and constituted more of an exercise in logical manipulation than in mathematics.

A demonstration of MIZAR-MSE was presented during the International Congress of Mathematicians, Warsaw 1982, in August 1983. The demo included a very nice example suggested by Prof. J. Łoś:

Prove that if the union of two equivalence relations is full then one of the relations is full.

An interesting experiment with MIZAR-MSE[4] was run in the popular mathematics and physics monthly *Delta* for 10 months starting in September 1983. For 10 consecutive months *Delta* printed short papers about MIZAR-MSE and this was intended to form a gentle course on the system. Each month three exercise problems were posted and the readers were encouraged to send in their solutions on paper by regular mail (as it was several years before the Internet). The solutions were typed in and checked by the machine and the results sent back to the readers by regular mail.

MIZAR-MSE was used in the preparation of a number of MSc theses, the first of which was by Henryk Oryszczyszyn titled *A generalization of the Szmielew oriented order (on dendrites)*.

9. 1982: MIZAR-3

MIZAR-3 was meant to be an extension of MIZAR-2 and its implementation was attempted on ODRA 1305 in Pascal-1900. It was the first multi-pass MIZAR processor with a lot of stress on the design of intermediate files. MIZAR-3 had a richer and more systematic syntax than MIZAR-2. Andrzej added some keywords for naming the various correctness conditions required for different definitions: **existence**, **uniqueness**, **coherence**, **consistency**, **correctness** and these keywords are still with us today. MIZAR-3 was based on the von Neumann-Bernays-Gödel set theory with classes.

This version of MIZAR was entirely experimental, never completed,[10] and never used for any substantial formalizations.

10. 1983–84: MIZAR-HPF

The language of MIZAR-HPF was designed by Andrzej in 1983 and then implemented on PDP-11 under RSX. This was the first time when working from a monitor became commonplace and a dedicated editor for this MIZAR was created. This editor, called EDH, provided syntactic checking for MIZAR-HPF and was designed and implemented by Stanisław Żukowski, who also implemented the reasoning (contents of proofs) checker. Further processing was designed only for syntactically correct texts. Both the editor and the processor proper were driven by a modifiable LL(1) grammar which facilitated experimenting with syntax. There were many such experiments, but not too many of them left a tangible trace. The semantic analyzer was written by Czesław Byliński and the inference checker by Andrzej Trybulec.

While MIZAR-HPF was meant as a sub-language of MIZAR-3, it is probably best seen as a collections of extensions of the frugal MIZAR-MSE. Features added include:

1. Unary and binary functions could be written in the usual prefix and infix notation but the functional notation of term constructors also included:
   - **the** F of $x_1, x_2, \ldots, x_n$, e.g., **the center of G**, **the line of a, b**.
   - Operational brackets (always in pairs: left and right), e.g., **[x,y]** for an ordered pair, **{x,y,z}** for a set.
   - General notation for functions $x_0(x_1, \ldots, x_n)$ where $x_0, x_1, \ldots, x_n$ are terms, e.g., **f(x,y)**, **(f*g)(x)**, **(f+g)(x)**.
2. A richer set of formats for atomic sentences:
   - Attribute format in general form x_1 **is** A **of** $x_2, \ldots, x_n$ with variants: x_1 **is** A, or $x_1, x_2, \ldots, x_k$ **are** A, or $x_1, x_2, \ldots, x_k$ **are** A **of** $x_{k+1}, \ldots, x_k$. E.g. **x is even**, **f is inverse of g**, **a, b are isomorphic**.
     The more natural format of negation for such sentences was also introduced, e.g., **x is not even**.
   - For binary predicates the infix format was provided.
   - A special format for ternary predicates, e.g. **x = y wrt E**
3. Sorts with parameters, in a general format T **of** $x_1, x_2, \ldots, x_n$, where T is an identifier (when $n = 0$ then **of** must be omitted). Parameterized sorts permit substantial economy in constructing terms. The standard illustration of this point is the composition of two morphisms in a category: consider the following declarations

[10] One of the unpleasant side effects of this effort was that overworked Czesław Byliński, the leading implementer, ended up in a hospital for several weeks.

```
let C      be category;
let a,b,c  be object of C;
let f      be morphism of a, b;
let g      be morphism of b, c;
```

A straightforward notation for the composition of two morphisms could have looked like **Comp(C,a,b,c,f,g)** where all six arguments of the composition had to be specified explicitly.

In MIZAR-HPF the composition could use the natural notation **f*g** where the remaining four arguments could then be reconstructed from the sorts of **f** and **g**. (This feature was available to some extent in MIZAR-2.) The omitted arguments were called *hidden parameters*, and thus HPF, for hidden parameters and functions.

4. Default quantifiers allowed for skipping of leading universal quantifiers at the formula level. Thus, instead of

```
for A, B, C being SUBSET of U st A <= B & B <= A holds A = B
```

one could shortly state **A <= B & B <= A implies A = B** to achieve the same result provided an appropriate predeclaration (reservation) was made for **A**, **B** and **C** earlier.

In 1984, Andrzej started his one-year visit at the University of Connecticut in Storrs.[11]

11. 1986–1988: MIZAR-4

In 1986, MIZAR-4 was implemented as a redesign of MIZAR-2, but taking into account features of all previous versions. The implementing team consisted of Czesław Byliński, Marcin Mostowski, Andrzej Trybulec, Edmund Woronowicz, Anna Zalewska and Stanisław Żukowski. Since the target machine, PDP-11, was relatively small it was necessary to design a number of passes that communicated through files. Originally there were seven passes and the split into passes was mainly forced by the limited amount of memory on the machine (which were really Soviet clones of PDP-11 named SM-4 working under RSX-11). Over time the passes were taking on meaningful roles and their number was reduced to four when MIZAR-4 gave birth to PC-MIZAR in late 1988.

In late 1986 MIZAR-4 was ported to PCs and distributed to several dozen users over the next few years (its distribution continued until early 1989). MIZAR-4 was also an experimental system that was subjected to intensive evolution.

The switch to PCs under DOS also resulted in a not very good decision to use extended ASCII IBM Set II. The initial excitement of having several dozen characters that frequently occur in mathematical texts (and many characters that do not occur there at all) faded very quickly in the first attempt to port the system to Linux in November of 1999. The extended ASCII disappeared in September of 2001, however, its 12 year presence caused a lot of grief for people that did not use vanilla DOS systems (but it did not concern the core MIZAR team).

We would like to mention that Grzegorz Bancerek started his university studies in 1985 and by 1987 joined the MIZAR group; Grzegorz's presence has had a big impact on the entire future of MIZAR.

[11] Most likely, this visit would have had continued if not for Andrzej's problems with his US visa. Having a single entry US visa, Andrzej left the US for a MIZAR workshop in Belgium and upon his return was stopped at the US border. But this story is best told by Andrzej himself.

11.1 Vocabularies

Unlike in all previous MIZARs, one could now define multi-character symbols to be used in formats of predicates and functors and one could also define operational brackets. These symbols constituted a separate lexical category and parsing relied on their recognition. The symbols were declared in special files called *vocabularies*.

The declaration of vocabulary symbols added substantial diversity to the MIZAR texts as now the allowed formats for predicates and operations included infix (prefix, postfix) notation with an arbitrary number of left and right (right, left) arguments. Although it does not seem like a big change, written MIZAR texts became more varied and easier to read. Further, within the **environ** part, one could define priorities for the defined symbols of predicates and functors such that some economy of parentheses was under control of the text author.

But there was also a side effect causing quite unexpected albeit minor troubles. Namely, an identifier declared as a symbol in a vocabulary could not serve as an identifier anymore. The most frequent mishap was with the single character **U** which was used as symbol for set union and any attempt to use it as an identifier (for a variable or a label) led to a not immediately obvious syntactic error. Even experienced MIZAR users tripped over this. It took years before such "symbolic" traps were eliminated (set union is now written as **\/**).

11.2 Predeclared

Two modes **set** and **Any** were predeclared but their meaning must have been given in every article. (Mode **Any** has been eliminated in mid 90s.) Also, modes **Element of** and **set of** were predeclared, the former took an argument which was a **set** while the latter's argument was a mode. The elementhood relation was not predeclared and some authors used **in** while other preferred $\in$, a character available in extended ASCII.

Nat was predeclared and small non-negative integer constants were recognized as objects of type **Nat**.

11.3 Reservation

One could **reserve** types of variables, for example:

```
reserve a,b,t,x,y,z,m,n,k for Nat;
```

such that later one did not have to specify types of variables

```
for t,x holds t*x=x*t;
```

and one could also omit leading universal quantifiers

```
n + m = m + n;
```

which were added automatically. Interestingly, there were authors who preferred not to use this feature.

11.4 Definitions

Definitions got uniform syntax and were written in definitional blocks delimited by **definition** and **end**. The keywords **func**, **pred** and **mode** indicated the nature of the defined notion. In definitions of functors, the keyword **it** was used to denote the object being defined, e.g.,

```
definition
  let A, B be set;
  func A u B -> set means
   Union:  x in it iff x in A or x in B;
```

The above defines the union of two sets and in the definiens **it** denotes the union of **A** and **B**. This keyword was also used in **mode** definitions, e.g.

```
definition
  mode open_set -> set means open: it = Int it;
 end;
```

The definitions of functors and modes required correctness conditions to be proved; existence and uniqueness for functors; existence (i.e., non emptiness) for modes.

One could change the type of a functor by using the **redefine** statement, e.g.,

```
definition
  let A be set;
  let X, Y be Finite_Subset of A;
  redefine X u Y as Finite_Subset of A;
end;
```

The redefined functor had a more specific type as its arguments had narrower types. For a redefinition one needed to prove that the redefined functor was coherent with the original.

11.5 Schemes

Schemes got a new syntax and the induction scheme was now written as

```
scheme IND {P[Nat]}:
for n being Nat holds P[n]
  provided
    P[0] and
    for n being Nat holds P[n] implies P[n + 1];
```

11.6 A problem

MIZAR-4 evolved and its evolution was influenced by formalizing in MIZAR-4 more and more of interesting mathematics. Stanisław Czuba maintained a collection called *CAMT* for *Central Archive of* MIZAR *Texts*.[12] At the end of 1988, this collection included 19 texts contributed during 1987 and 1988.

It was easy to notice that the developed texts overlapped a lot especially in the environment part where authors were stating set theoretical preliminaries over and over again. Forever, people's tastes varied and these set theoretic preliminaries were stated using different notation. This led to a lot of repeated efforts and thus a waste of resources. Some sort of communication between independently developed texts was needed. Before this happened, MIZAR as a project got a financial boost.

[12] In Polish: *CATM* for *Centralne Archiwum Tekstów Mizarowych.*

12. 1987: RPBP III.24

RPBP III.24 was not a name of a version of MIZAR. It is an acronym of a state research grant program of the Polish Ministry of Science and Higher Education from which the MIZAR group obtained substantial financing for a project named *Logical systems and algorithms for computerized checking of proof correctness* where the main goal of the project was stated as

> Solving the problem of whether or not there is a system of logic suitable for
> – formalization of mathematical texts without substantially increasing their size, and
> – automated checking of their correctness.
> In particular, in answer to the question of whether or not and if so then to what degree, the Polish system MIZAR satisfies the above requirements and determining directions of its development and its scope of application.

The research program was coordinated by Prof. Witold Marciszewski and lasted for five years, 1987-1991. The grants obtained through the program provided major funding for the development of the MIZAR system and especially the library. During these five years, a dozen of scientific institutions from all over Poland were involved with the participation of about 100 people. These efforts resulted in almost 250 MIZAR articles being contributed to the MIZAR library.

13. 1988-89: MIZAR and MML

The ongoing evolution of MIZAR-4 and its implementation on PCs, prompted Andrzej to name the language simply MIZAR and call its implementation PC-MIZAR. While articles in previous versions of the language must have been self-contained, the final MIZAR has an accompanying data base and allows for cross-references among articles. The role of the environment part of an article has changed from that in MIZAR-4: the environment section may now only contain directives importing resources stored in the data base.

The implementation of PC-MIZAR was carried out during 1988 by Andrzej with Czesław Byliński and other implementors of MIZAR-4. The first three articles were included into the data base on January 1, 1989—this is the official date of starting the Mizar Mathematical Library—MML, although this name appeared much later.

13.1 Axiomatics

As of January 1, 1989, the MIZAR data base consisted of three axiomatic articles contributed by Andrzej:

– **HIDDEN**, see [8, pp. 191–193], contained the declarations of built-in notions and there were quite a number of them:
  - modes: **Any**, **set**, **Element of**, **DOMAIN**, **TUPLE of**, **Subset of**, **SUBDOMAIN of**, **Real**, **Nat**.
  - predicates: $=$, $\in$, $\leq$ (for real numbers).
  - functors: Cartesian product of 2, 3, and 4 sets, **bool**—powerset, **REAL**—the domain of real numbers, **NAT**—the domain of naturals, **+** and $\cdot$ for addition and multiplication of reals.

  This special article has been substantially trimmed over time, see [3] and now contains only the declarations of: mode **set**, equality **=** and inequality **<>** and elementhood, now written as infix **in**. All other elements originally in **HIDDEN** have been constructed and are not built-in anymore.

– **TARSKI**, see [11], contained axioms of essentially ZF set theory in which the axiom of infinity was replaced by the axiom of existence of arbitrarily large, strongly inaccessible cardinals (the so called Tarski axiom). This choice of foundation instead of just ZFC was motivated by certain constructions done in category theory.
  This axiomatic article has changed only a little, see [12]. One change concerned the removal of the auxiliary mode **Any**, the other change removed the axiomatic definition of powerset as it can be properly defined using the Tarski axiom.
– **AXIOMS**, see [13], titled *Built-in Concepts*, provided properties of built-in notions declared in **HIDDEN** including axioms of strong arithmetic of real numbers and naturals.
  Having the real numbers available axiomatically from the beginning allowed users to develop a lot of mathematics right away. It was as late as March 1998 when Andrzej and Grzegorz Bancerek completed the construction of real numbers. The axiomatic article **AXIOMS** became a normal article in which all theorems are proven and its title is now *Strong arithmetic of real numbers* (see [14]).

13.2 MML

The first "regular" article titled *Boolean Properties of Sets*, [16] was included into the data base on January 6. By the end of 1989, there were 66 articles in the collection, covering mainly the basic mathematical toolkit. 15 years later, at the time of this writing, MML consists of 855 articles and about 50 articles are contributed each year. Almost all of the material about the current state of MML is available at **http://mizar.org**.

MIZAR—the language—is a formal system of general applicability and as such has little in common with any set theory. MML is a specific application of MIZAR in developing mathematics based entirely on a set theory (see [12]). In building MML, the MIZAR language, the assumed logic, and the chosen set theory provide an environment in which all further mathematics is developed. This development is *definitional*: new mathematical objects can be defined only after supplying a model for them in the already available theories. MIZAR has been always based on classical logic because this is the logic used by almost all mathematicians.[13]

In the last 15 years, the evolution of the MIZAR system has been driven by the growth of MML. All major components of the system—the MIZAR language, the verifier and the MML itself—have undergone numerous changes, too many to even mention the more important ones here. Unfortunately, there is only sparse and incomplete documentation that would allow tracing this evolution. The situation has improved substantially since all the MML files have been kept under CVS from the beginning of 2002.

Although the MIZAR language and its processor evolve, their pace of change is relatively slow. However, even a small change in the language or the verifier may cause vast changes in the MML; MML is maintained to conform with the current version of the processing software. Besides these types of updates, MML undergoes many changes and it is continually revised in an effort to improve its integrity. This imprecise term covers quite a number of diverse issues: choice of symbols and notation for new (and old) constructors, typing hierarchy, repetition or presence of almost equivalent notions, redundant (repeated or weaker) theorems, cumbersome formulation of theorems, etc.

MML is centrally maintained by the *Library Committee* now headed by Adam Grabowski. Essentially every submitted article accepted by the MIZAR verifier and which passed some automatic review is included into the library. Over the years, MML has grown to the size that searching it has become a problem for MIZAR authors. Even if one knows a lot of the library by

[13] Let us note that most of the other proof assistants are based on some other logical calculi.

heart, one still would like to use a tool for finding a needed fact or notion. For many years the search tool of choice was the **grep** utility. However, because MIZAR uses overloaded notations and different authors do not use notations in a uniform fashion, such searches usually return a lot of irrelevant material while missing some relevant items. Since 2001 Grzegorz Bancerek has been developing a tool called MML QUERY which allows for semantics based browsing and searching.

The organization of material stored in MML is not fixed. MML can be seen as a collection of intertwined MIZAR articles where authors include whatever is to their liking. Such articles correspond to *primary* scientific information that over time give rise to the *secondary* information, i.e., overviews, monographs, textbooks. In 2002, the MIZAR team has begun building an Encyclopedia of Mathematics in MIZAR, EMM, whose articles have a monographic character and are extracted from the "raw" material of the contributed articles in a *semi-automatic* way. This process is assisted by MML QUERY. Currently, there are five such encyclopedic articles covering: Boolean properties of sets and basic arithmetic of real and complex numbers.

13.3 FM and JFM

In April-May 1989, Andrzej visited Edmonton and together with the second author was working on a new MIZAR article. This work revealed the need for a better means to search the available articles, or at least to browse through and read them. A superficial translation of MIZAR articles into TEX was prepared and reported in [8]. This small experiment was one of the steps toward the printed journal *Formalized Mathematics*[14]. Several months later Roman Matuszewski and Stanisław Żukowski prepared more sophisticated technology for translating MIZAR abstracts into stilted English and then typesetting them in TEX (this work was initially supported by Fondation Philippe le Hodey, Bruxelles).

The problem with *Formalized Mathematics* is that once printed on paper the published abstracts have only some archival value and they do not reflect the evolving nature of MML making them of limited use for MIZAR authors. In the age of Internet, it was quite natural that an electronic, hyper-linked version of MIZAR abstracts be created. This materialized in years 1995–97 when *Journal of Formalized Mathematics*[15] was created with support from Office of Naval Research, USA. This electronic journal reflects the current version of MML abstracts in a variety of formats.

References

1. A. Grzegorczyk. *Zarys arytmetyki teoretycznej* (in Polish). PWN Warszawa, 1971.
2. L. A. Kalużnin. *O języku informacyjnym matematyki* (in Polish). Wiadomości Matematyczne, Roczniki PTM, VII(2), Warszawa 1964. This paper originally appeared in a collection: *Prikladnaja ligvistika i mashinnyj perevod*, Izdatielstvo Kievskogo Universiteta, 1962, pp. 21–29.
3. Library Committee. MIZAR Built-in Notions. **http://mizar.org/JFM/Axiomatics/hidden.abs.html**.
4. K. Prażmowski and P. Rudnicki. MIZAR-MSE. Delta *(monthly, in Polish)*, 1983, **10**(9), pp. 8–9. *This is the first in a series of 10 popular articles which appeared monthly until June, 1984.*
5. E. Ramm and E. Woronowicz. *A computerized system of proving properties of programs.* ICS PAS Reports No. 403, March 1980, Warszawa.
6. P. Rudnicki. What should be proved and tested symbolically in formal specifications? In *4th IEEE International Workshop on Software Specification and Design*, pages 190–195, Monterey, Ca., 1987.
7. P. Rudnicki and W. Drabent. Proving properties of Pascal programs in MIZAR-2. *Acta Informatica*, 22:311–331, 1985. Erratum pp. 699–707.

[14] **http://mizar.org/fm**
[15] **http://mizar.org/JFM**

8. P. Rudnicki and A. Trybulec. A Collection of TEXed Mizar Abstracts. University of Alberta, Dept. of Comp. Sci., 1989, TR 89–18.
http://mizar.org/project/TR-89-18.ps
9. A. Trybulec Informationslogische Sprache MIZAR Schrifterreihe des Institutes für Informationswissenschaft, Erfindungswesen und Recht, Technische Hochschule Ilmenau, Heft 33, Ilmenau 1977.
10. A. Trybulec. J ezyk informacyjno-logiczny MIZAR-MSE (in Polish). ICS PAS Reports 465, March 1982.
11. A. Trybulec. Tarski Grothendieck set theory. *Formalized Mathematics*, 1(**1**):9–11, 1990.
12. A. Trybulec. Tarski Grothendieck set theory.
http://mizar.org/JFM/Axiomatics/tarski.html
13. A. Trybulec. Built-in Concepts. *Formalized Mathematics*, 1(**1**):13–15, 1990.
14. A. Trybulec. Strong arithmetic of real numbers.
http://mizar.org/JFM/Addenda/axioms.html
15. A. Trybulec. A Borsuk Theorem on Homotopy Types. *Formalized Mathematics*, 2**4**:535–545, 1991.
http://mizar.org/JFM/Vol3/borsuk_1.html
16. Z. Trybulec and H. Święczkowska. Boolean Properties of Sets. *Formalized Mathematics*, 1(**1**):17–23, 1990.

MECHANIZED MATHEMATICS AND ITS APPLICATIONS, VOL. 4, NO. 1, MARCH 2005, 25:33

On the Computer-Checked Solution of the Kuratowski Closure-Complement Problem

Adam Grabowski

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

Abstract – Since its beginnings back in 1989, the development of the Mizar Mathematical Library has been strongly stimulated by the formalization of general topology. In this paper we describe a solution of the famous Kuratowski closure-complement (and, as a side-effect, the closure-interior) problem, which are more a kind of mathematical puzzle (or exercise for students) than regular mathematics. Based on the large Mizar Mathematical Library, especially using its well-developed part devoted to the topology, we also construct examples in the topology of a real line illustrating both exercises. We were surprised by the influence of our solution of this problem on the improvement of the whole Mizar library.

1. Introduction

Mizar is considered to be a hybrid of (at least) three elements: the first part is the language which was developed to help mathematicians writing their papers in a form understandable by machines; the second is software for checking correctness and collecting the results of this work and last – but definitely not least – is the Mizar Mathematical Library which is considered the largest repository of the computer-checked mathematical knowledge in the world. While the starting date for the idea of Mizar as it is fuzzy believed is 1973/74, the beginnings of MML are exactly stated – January 1, 1989. So the year we celebrate 30 years of Mizar is also the 15th birthday of MML.

In the beginning, the development of MML was rather an experiment of how to deal with the system to make it usable not only for computer scientists, but also for mathematicians to assist them in their ordinary everyday work, now its primarily aim is to collect knowledge in a uniform way close to mathematical vernacular, checkable by computers as well as readable for humans. Clearly then, such large repositories can also be a kind of a handbook (or a set of handbooks) for students.

To avoid making Mizar a fossilized mathematical Latin, but instead a living language useful for contemporary scientists, it is quite natural to pay special attention to the disciplines which are most promising, that is which have great impact on the whole of mathematics. It is clear however that one cannot formalize brand new results without a properly done background which usually has rather archival flavour. However the formalization of older papers can shed new light on them or propose new approaches to the well-known notions.

Since its beginnings back in 1989, the development of the Mizar Mathematical Library has been strictly connected and stimulated by the formalization of topology. Now this tight connection is by no means over: the Białystok team is working to reach completion (at least we all hope so) of the Jordan Curve Theorem proof, Mizar formalization of the *Compendium of Continuous Lattices* (CCL) still forces the MML developers to revise previous articles in the topic, and work on the proof of the Brouwer fixed point theorem has started recently.

Manuscript received August 17, 2004; revised October 22, 2004

As we have mentioned earlier, the notion of a topological space needed some preliminary development, such as basic properties of sets and subsets, functions and families of subsets of a set. Three months after the start of MML, a Mizar article (only 27th in the collection) defining the notion of a topological space was accepted, namely **PRE_TOPC** [10] dated on April 14, 1989. This proves that topology is a nice subject for formalization, being also one of the main lectures for students with many good textbooks (but also with many different approaches) to follow. Due to its many independently developed subtopics it can be formalized rapidly by a large group of people. One of the irrefutable arguments also was that Andrzej Trybulec who is the designer of Mizar holds a PhD in topology.

What should be mentioned here is also a didactic value of this formalized discipline: five exercise sets in topology were prepared for PC Mizar by Czuba and Bajguz (Białystok) in 1989–90, ca. 100 pages each. A similar choice for didactic experiments (but without Mizar usage) was made for instance described by Cairns and Gow [3].

The paper is organized as follows: in Section 2 we briefly introduce the problem of 14 Kuratowski sets, Section 3 presents the formalization of the elementary topological notions we used. Section 4 presents how the problem was solved, while in the last section we draw some concluding remarks.

2. Fourteen Kuratowski Sets

The primary formulation of this problem was stated by Kuratowski in 1922 [8]:

> *La table (T) (p. 186) renferme 14 ensembles que l'on peut obtenir d'un ensemble A donné en combinant les opérations $\overline{A}$ et A'. En s'appuyant sur l'axiome IV, le théoréme 6 et le principe de la double négation ($A'' = A$), on montre aisément que le nombre de 14 ensembles ne pourrait être augmenté. Nous allons montrer qu'il ne peut être diminué nonplus.*

The table (T) from the above text is reproduced in Fig. 1.

> *If A is a subset of a topological space X, then at most 14 sets can be constructed from A by complementation and closure. There is a subset of the real numbers (with the usual topology) from which 14 different sets can be so constructed.*

The problem which was advertised by Kelley in [7] in the above form, is a particular case of the following theorem:

> *Let A be an ordered set and let $f : A \longrightarrow A$ be an increasing, expanding and idempotent mapping, and let $g : A \longrightarrow A$ be a decreasing involution. Then, the semigroup generated by f and g consists of 14 elements (at most).*

Along these lines, we will use A' to denote a complement of A and A^- instead of a closure $\overline{A}$ to make notation more readable.

A full proof of the problem consists of the following two parts:

- no more than 14 sets can be obtained,
- there exists an example of a topological space and its subset A so that exactly 14 different sets are produced by applying the closure and complement operators.

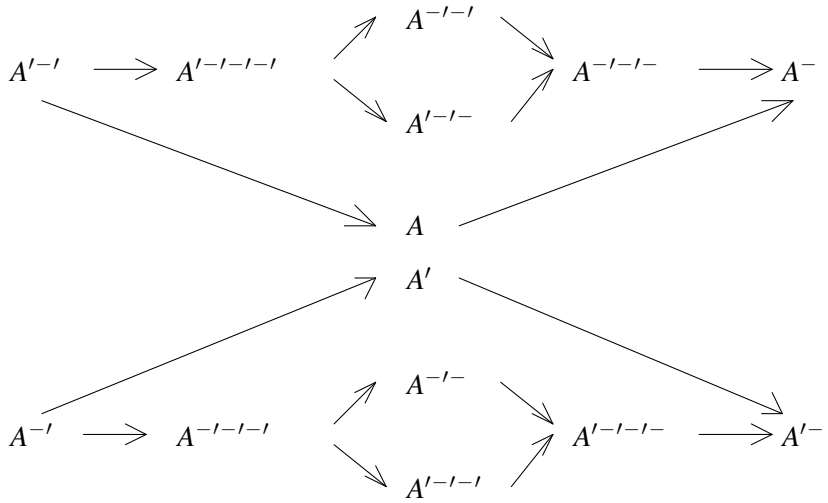

Figure 1. Fourteen Kuratowski sets ordered by inclusion

Considering only the upper (lower) part of the diagram we can illustrate another Kuratowski problem: the maximal number of different sets we can obtain by applying the interior and closure operators to any subset A of an arbitrary topological space is seven. Although both problems of Kuratowski type are of rather elementary character, we aimed at the possible elegant and maximal reuse of Mizar features. The first approach to the formalization was written in 1996 by the author, but the construction of an example of a concrete topological space and subset which generate all 14 sets (7 sets, respectively) seemed too work-consuming in those days.

Continuous work on improving the implementation of the Mizar system changed the views for certain details of the proof. We mind mainly the extension of syntax for brackets which can be paired in different ways (e.g., left-open right-closed intervals, etc.), the removal of the notion of the so-called unclusterable attributes and further development of MML, especially preparatory articles for the proof of the Jordan Curve Theorem.

A Mizar article (over 1500 lines) containing the complete formalized solution by Bagińska and Grabowski [2] was a part of Bagińska's B.Sc. thesis in mathematics. It was an interesting experience because the student had not worked with the Mizar system before.[1]

3. Mizar Formalization of Topology

In this section we will briefly introduce the state of the formalization of the topology in the MML. The main facts which are encoded in MML are shown below (we also cite the MML identifiers of the articles which contain the results):

- Tichonov theorem (**YELLOW17**),
- Fashoda meet theorem (**JGRAPH_1**),
- Urysohn lemma (**URYSOHN3**),
- Jordan Curve Theorem – a version for special polygons (**GOBRD12**),
- Brouwer fix point theorem for real intervals (**TREAL_1**),
- Stone–Weierstrass theorem (**WEIERSTR**),
- Nagata–Smirnov theorem (**NAGATA_2**).

[1] In fact, she soon left the project and she is not working with Mizar as of now.

3.1 Structures and Attributes

The backbone structure for the entire topological part of MML is the immediate successor of **1-sorted**, that is **TopStruct** (extended only by the selector **topology** which is a family of subsets of the carrier. Originally it was designed as a type with non-empty carrier, so many of the theorems are still formulated for **non empty TopStruct**. While there are over 100 structure types in the whole MML, here the formal apparatus is very modest and restricted only to this one type. Properties of a topological space are added to it through the attribute mechanism, namely **TopSpace-like**. Since it is very natural (the carrier of the space belongs to the topology, it is closed also for arbitrary unions of elements and binary intersections), we will not cite it here. This adjective is used in the definition of the Mizar type to understand **TopSpace** simply as **TopSpace-like TopStruct**.

3.2 Basic Notions

As mentioned earlier, the main Mizar functors used for our purpose were closure (which is not introduced in MML by four Kuratowski axioms, but is of the form below) and complement (defined naturally for a subset A of E as $E \setminus A$).

```
definition let GX be TopStruct, A be Subset of GX;
  func Cl A -> Subset of GX means   :: PRE_TOPC:def 13
  for p being set st p in the carrier of GX holds p in it iff
    for G being Subset of GX st G is open holds
      p in G implies A meets G;
end;
```

We will use a new postfix synonym for **Cl**, namely **−**.

The attribute **open** describes elements which belong to the topology of a given space. As a standard, a subset is **closed** if its complementation is **open**.

3.3 Some Statistics

About one-fifth of the Mizar library (i.r., 167 out of 855 articles in MML, i.e. about one fifth) deals with topology. Even if calculation is rather rough since there is no classification of MML e.g., according to 2000 AMS Subject Classification (and criteria are not exactly clear; as a rule we counted articles which used any notation from [10], [4] or those dealing with formal topology), it is a significant amount.

Table 1. Statistical data about topological articles

Article series	Number of articles	kB	Subject
TOPS	4	168	basics of general topology
TOPREAL	9	736	Euclidean topological spaces
GOBOARD	14	1556	Go-Boards and special polygons
JORDAN	36	4202	JCT core series
JGRAPH	6	2019	versions of the Fashoda theorem
BORSUK	6	530	products, algebraic topology
WAYBEL	24	1735	CCL-needed stuff
FINTOPO	4	163	formal topology
others	64	4154	
Total	167	15263	

There is an advanced searching tool – MML Query – which is good for authors who are rather well acquainted with Mizar (and behaving better when searching, not when browsing), but due to requests obtained from users to gather articles in a Bourbaki manner we plan to add AMS classification manually in the nearest future. We are considering adding the mandatory field to the bibliography file as AMS 2000 to be filled by the author but we will have to classify 855 articles by hand.

Some measurements of information flow between articles have been done by Korniłowicz. The amount of information that an article A transfers to an article B is calculated as the sum of information transferred by all theorems from A which are referred to in B, counted according to the Shannon formula. The article A is a direct ancestor of B if it transfers the largest quantity of information into B. Using this criterion, 32 articles are descendants of **PRE_TOPC** in a tree of dependence of all MML items.

Most of "topological" articles are devoted to the proof of the famous Jordan Curve Theorem. The first article dedicated to this topic is dated at the end of 1991. Even if in Mizar there is only a version of JCT for special polygons fully proved, the theorem has had a great impact stimulating the development of MML. Statistically, more than five articles devoted to this topic were submitted yearly (three in a year if we count only **JORDAN** series), basically written by the authors of University of Białystok, Poland and Shinshu University, Nagano, Japan.

3.4 Topology of a Real Line

To construct a real example we had the possibility of choosing between the two representations of a real line with the natural topology. We decided to use **R^1** which is the result of an application of a functor **TopSpaceMetr** on the metric space **RealSpace**.

Due to the lack of space we will not explain in detail the differences between the three representations of a natural topology on the real line: **REAL**, **R^1** and **TOP-REAL 1**, but developing in parallel two approaches to the pairs: the ordinary ordered pair as defined by Kuratowski and the two-elemented finite sequence seems too ineffective.

4. The Solutions

In this section, we present how the problem of fourteen Kuratowski sets was expressed in the Mizar language. Although reusability of this fact is rather small, the problem itself is well-known in the literature and deserves to be formalized in MML. Even if many applications in other terms (e.g., group theory) are available, we decided to code it in a clean topological language.

4.1 The Closure-Complement Problem

When we tried to formulate the whole problem in Mizar, at the beginning we met one of the low quantitive restrictions of the Mizar language: enumerated sets can have at most ten elements. This constant can be easily extended, but without changes in the checker itself, the division of Kuratowski sets into two smaller parts with cardinality seven appeared to be very feasible.

When thinking of the partition of the Kuratowski sets, one can imagine a few classifications: one of them is for example to distinguish those where variables occur in positive (i.e., not complemented) or negative form. This is reasonable when we look at the diagram of Fig. 1. The positive part is the upper one.

The most useful approach however, as we point out later on, was to divide the family of subsets into its closed and open parts, and the rest. Thanks to the functorial clusters[2] mechanism we obtain immediately, that the closure of a set is closed, and the complement (` in Mizar) of a closed set is open.

```
definition let T, A;
  func Kurat14ClPart A -> Subset-Family of T equals :: KURATO_1:def 3
    { A-,  A-`-,  A-`-`-,  A`-,  A`-`-,  A`-`-`- };
  func Kurat14OpPart A -> Subset-Family of T equals :: KURATO_1:def 4
    { A-`, A-`-`, A-`-`-`, A`-`, A`-`-`, A`-`-`-` };
end;

theorem :: KURATO_1:5
  Kurat14Set A = { A, A` } \/ Kurat14ClPart A \/ Kurat14OpPart A;

registration let T, A;
  cluster Kurat14Set A -> finite;
end;
```

The registration above is to enable the use of the functor **card** yielding natural numbers. This is needed to allow a comparison with numerals, because essentially the **Card** functor, of which **card** is a synonym, yields ordinal numbers. The following lemma was a rather easy provable property of the Fraenkel operator.

```
theorem :: KURATO_1:7
  card Kurat14Set A <= 14;
```

To prove that the 14-Kuratowski set is closed for the operations, we effectively used only a fact that $A^{-\prime-\prime-\prime-} = A^{-\prime-}$ (or **A-`-`-`- = A-`-** in Mizar notation). As it turned out to be convenient, $A^{--} = A^{-}$ should be introduced as a **projectivity** property in Mizar, similarly as it is in case of a complement operator (**involutiveness**). The tactic of using **per cases** was also very useful and intuitive. To conclude that both operators do not map outside the given set, we formulated and proved the following theorem.

```
theorem :: KURATO_1:6
  for Q being Subset of T st Q in Kurat14Set A holds
    Q` in Kurat14Set A & Q- in Kurat14Set A;
```

The generation of all 14 Kuratowski sets can be treated as the result of a certain algorithm, so it would be enough to prove that this algorithm terminates. A similar problem occurred in the case of the famous elegant Rado proof of the Hall Marriage Theorem [11] when we had to formalize the process of decreasing the cardinality of any element of the family of subsets. Because as of now, there is no comfortable method of algorithm coding in Mizar, we had to handle it in a different way.

```
registration let T be non empty TopSpace, A be Subset of T;
  cluster Kurat14Set A -> Cl-closed compl-closed;
end;
```

The above cluster uses new attributes, which can be translated (as in fact they were in the *Journal of Formalized Mathematics* [1] automatically) as “closed for closure operator” and “closed for complement operator”. The registration has its justification based on **KURATO_1:6**, which completes the proof of the first part.

[2] Bancerek proposed *term adjective registration* for this.

Using methods developed and checked in the case of the closure-complement problem, we were able to solve analogously the other problem of Kuratowski, that is closure-interior. The solution was even less problematic because of the smaller number of sets (below the 10-arguments barrier).

4.2 The Example of 14-Subset

It is natural to search for an appropriate example in the real line with the natural topology, as Kuratowski proposed. Frankly speaking, the lemmas for the real euclidean plane are developed much better. We decided to use the set as shown below.

```
definition
  func KurExSet -> Subset of R^1 equals :: KURATO_1:def 6
    {1} \/ RAT (2,4) \/ ]. 4, 5 .[ \/ ]. 5,+infty.[;
end;
```

To shorten the notation we had to define intervals of rational numbers (only open, as **RAT(a,b)**). It is worth mentioning here that to keep notations for intervals as compact and close to the mathematical tradition as possible, we introduced **].a,+infty.[** as a synonym to the previously defined in MML **halfline(a)**. Both functors have only one argument since obviously **+infty** is not an element of set of reals. For this part of the proof the predicate for an enumerated list of arguments, namely **are_mutually_different**, was also essential.

Due to the lack of decimal fractions in MML, the calculus of intervals of numbers was not comfortable. We had to state for example a theorem of the density of the irrational numbers, even the formalization of the proof of the irrationality of the base of natural logarithms is relatively new. The fact of the irrationality of the square root of 2, which was advertised well by Wiedijk in his famous comparison of 15 provers of the world, was also useful. Too few examples were previously done by others – but MML focuses on the reusability of introduced theories. The policy, which resulted in a low number of exercises formalized during the CCL project, caused the necessity of the trivial calculus on intervals in [5].

Following [12], to show the mutual difference of subsets for each of the fourteen sets we may create a vector of boolean values 0's and 1's indicating whether the certain test points lie in the set or not. We can order these binary numbers by means of their decimal representation. As Rusin stated, although $2^4 > 14$, the number of four points is insufficient to show that all 14 sets are different. Table 2 presents points incidence where the test points were respectively: $1,2,3,4,5,6$. In our case all 6 points are needed. We used this good-looking way only partially because there is no comfortable method for representing it in Mizar.

Table 2. Incidence of points in the example set

$$A = \{1\} \cup \big((2;4) \cap \mathbf{Q}\big) \cup (4;5) \cup (5;+\infty).$$

$A = [1,0,1,0,0,1] : 41$	$A' = [0,1,0,1,1,0] : 22$
$A^- = [1,1,1,1,1,1] : 63$	$A'^- = [1,1,1,1,1,0] : 62$
$A^{-\prime} = [0,0,0,0,0,0] : 0$	$A'^{-\prime} = [0,0,0,0,0,1] : 1$
$A^{-\prime-} = [1,1,0,0,0,0] : 48$	$A'^{-\prime-} = [0,0,0,1,1,1] : 7$
$A^{-\prime-\prime} = [0,0,1,1,1,1] : 15$	$A'^{-\prime-\prime} = [1,1,1,0,0,0] : 56$
$A^{-\prime-\prime-} = [0,1,1,1,1,1] : 31$	$A'^{-\prime-\prime-} = [1,1,1,1,0,0] : 60$
$A^{-\prime-\prime-\prime} = [1,0,0,0,0,0] : 32$	$A'^{-\prime-\prime-\prime} = [0,0,0,0,1,1] : 3$

4.3 The Role of Attributes

In order to avoid $\frac{14 \cdot 13}{2} = 91$ comparisons to check that all 14 sets are different, we decided to use the adjectives machinery available in Mizar. The following lemma was crucial to reduce the complexity of the second part of the proof.

```
theorem :: KURATO_1:60
  for F, G being with_proper_subsets with_non-empty_elements
    Subset-Family of R^1 st
      F is open & G is closed holds F misses G;
```

Recalling that **Kurat14OpPart A** is collectively open (that is, all its elements are open subsets) and **Kurat14ClPart A** is closed, and taking into account that all elements of these families of subsets are both proper and non-empty (all these properties are automatically added via functorial cluster mechanism), we obtain a significantly reduced number of checks (which were done according to the Rusin's method). Additionally, because we know that **KurExSet** is neither closed, nor open, the final calculation is just summing the cardinalities of disjoint six-elemented sets and a doubleton.

5. Conclusions

The main achievement of this work is the Mizar article [2] which contains the formalization of the Kuratowski problem, its full solution in terms of clean general topology and the construction of an example of a set generating exactly fourteen Kuratowski sets. We used this method to solve also the Kuratowski closure-interior problem. We put the stress not on the brute force proving, but we tried to find a possibly elegant proof. As a by-product, we obtained the characterization of subcontinua of a real line with the usual topology [5].

Even if the problem itself has been solved completely, during the codification a number of ideas arose. Essentially, at the second stage of solving Kuratowski's problem, when the example set had to be constructed, it soon became apparent that the Library Committee of the Association of Mizar Users should consider a massive revision of MML to merge both metric and topological spaces, because without such a revision one has to recode between different approaches. We have three representations of the real line with the natural topology, and this revision should simplify the formal apparatus.

As a future work, we may point out the classification of sets, as proposed by Kuratowski (open and closed domains) which was improved for example in [6] (supercondensed and subcondensed sets). Some preliminaries are yet done in Mizar by Karno. Recent projects to split MML into concrete (or classical) and abstract parts (i.e., using the notion of a structure) have obviously one significant value: in this way some accidental connections between articles may be removed to make MML more compact. The larger items in MML are, the smaller the mess in the library – and consequently knowledge can then be retrieved more effectively. Topology is now clusterized relatively well; it turns out to be the most promising discipline to be the most thorough developed in MML.

Acknowledgements

Many thanks are due to anonymous referees for their remarks which enabled us to improve the paper, especially the suggestion to introduce a synonymous postfix notation **–** for closure operator **Cl** which significantly changed the readability of the Mizar article.

References

1. Journal of Formalized Mathematics, accessible at **http://mizar.org/JFM/**.
2. L.K. Bagińska and A. Grabowski, *On the Kuratowski closure-complement problem,* Formalized Mathematics, 11(3), 2003, pp. 321–327, MML Id in [1]: **KURATO_1**.
3. P. Cairns, J. Gow, and P. Collins, *On dynamically presenting a topology course,* Annals of Mathematics and Artificial Intelligence, 38(1-3), 2003, pp. 91–104.
4. A. Darmochwał, *Families of subsets, subspaces and mappings in topological spaces*, Formalized Mathematics, 1(2), 1990, pp. 257–261, MML Id in [1]: **TOPS_2**.
5. A. Grabowski, *On the subcontinua of a real line,* Formalized Mathematics, 11(3), 2003, pp. 313–322, MML Id in [1]: **BORSUK_5**.
6. Y. Isomichi, *New concepts in the theory of topological space – supercondensed set, subcondensed set, and condensed set,* Pacific Journal of Mathematics, 38(3), 1971, pp. 657–668.
7. J.L. Kelley, *General Topology,* van Nostrand, 1955.
8. K. Kuratowski, *Sur l'opération $\overline{A}$ de l'Analysis Situs*, Fundamenta Mathematicae, 3, 1922, pp. 182–199.
9. K. Kuratowski, *Topology*, PWN – Polish Scientific Publishers, Warszawa 1966.
10. B. Padlewska and A. Darmochwał, *Topological spaces and continuous functions,* Formalized Mathematics, 1(1), 1990, pp. 223–230, MML Id in [1]: **PRE_TOPC**.
11. E. Romanowicz and A. Grabowski, *Hall Marriage Theorem,* to appear in Formalized Mathematics, MML Id in [1]: **HALLMAR1**.
12. D. Rusin, posting available at **http://www.math.niu.edu/~rusin/known-math/94/kuratowski**.
13. E.W. Weisstein et al., *Kuratowski's Closure-Complement Problem,* from MathWorld – a Wolfram Web Resource. **http://mathworld.wolfram.com/KuratowskisClosure-ComplementProblem.html**.

MECHANIZED MATHEMATICS AND ITS APPLICATIONS, VOL. 4, NO. 1, MARCH 2005, 35:42

Mizar as a Tool for Teaching Mathematics

Krzysztof Retel and Anna Zalewska

University of Bialystok,
Institute of Computer Science, Białystok, Poland
retel@uwb.edu.pl
zalewska@hum.uwb.edu.pl

Abstract – The Mizar system is a proof–assistant based on classical logic. A substantial body of Mizar texts (the Mizar Mathematical Library) has been developed in various areas of mathematics. It has been quite natural to consider using Mizar as a tool for teaching mathematics; the experience in this matter dates back to 1975/76.

In this paper we present some historical facts about applications of Mizar in education. We also discuss the organization of a course on formalization of mathematics offered by the Institute of Computer Science at University of Bialystok. This course employs Mizar as the main tool of instruction.

1. Introduction

The main goal of the Mizar project[1] is to create a system for computer–aided formalization of mathematics. The ongoing development of the project has resulted in a language for recording mathematics and software for checking the correctness of written texts. The essential achievement of the project is a centrally maintained library of mathematics (Mizar Mathematical Library). The Mizar language (designed over many years by A. Trybulec) is a language suitable for the practical formalization of mathematics. The language is based on first–order logic and proofs are written in a style of natural deduction as proposed by Jaśkowski ([3]). During the development of the Mizar system the author conducted many experiments which guided the process of Mizar design. Some of these experiments concerned education.

The first use of Mizar as a tool for teaching mathematics dates back to 1975–76. The first implementation of the Mizar processor was used to teach propositional logic at the Płock Branch of Warsaw University of Technology (by Roman Matuszewski) and the Institute of Library Science and Scientific Information at Warsaw University (by Andrzej Trybulec).

An interesting correspondence course based on Mizar–MSE ([8]) was run for 10 months (September 1983 through June 1984) by a popular Polish science monthly *Delta*, a magazine aimed at secondary school students. An introduction to logic using Mizar–MSE (by P. Rudnicki and K. Prażmowski) was split into 10 parts and each month a part was published in *Delta* ([6]). Every part was furnished with a set of exercises the solution of which were meant to be checked by the Mizar–MSE checker. At that time, hardly anyone had free access to a computer and therefore the *Delta* readers were encouraged to send in their solutions of the exercises to the editor by regular mail. The solutions were then typed in and checked, and the results were sent back to the authors. The turnaround time for an exercise was rather long, around three weeks. Altogether the authors of the course received 241 solutions ([5]).

Starting from the academic year 1985–86, Mizar–MSE had been used to teach elementary logic in a number of one semester courses at the Bialystok Branch of Warsaw University at

[1] **http://mizar.org**

Manuscript received August 13, 2004; revised September 10, 2004

several departments: mathematics, education and physics. The principal goal of logic training was to develop skills in deductive reasoning ([9]).

In the spring semester of 1985, a course in foundations of geometry was taught at the Department of Mathematics of Warsaw University and it also employed Mizar–MSE ([7]). The course was based on the textbook [4] covering the formal exposition of axiomatic Euclidean geometry. However, Mizar–MSE did not support functional notation, so the use of the system had to be limited to chosen parts of the book. Carefully chosen theorems with an appropriate environment (stating the available facts) were assigned to students for proving. Students were expected to develop proofs in the Mizar–MSE language and to verify their correctness by the computer. The main goal of the course was to develop skills in proving simple theorems in geometry.

In 1987–88 a much richer version of Mizar, called Mizar–4, was applied in teaching "Introduction to mathematics" and "Lattice theory" at the Bialystok Branch of Warsaw University.

In the early 90s, PC–Mizar was used "indirectly" in teaching topology. For technical and organizational reasons, the laboratory classes were not possible to be conducted at that time. However, five scripts ([1]) with exercises and their solutions (written in the Mizar language and checked by the computer) were created and they were used intensively in teaching topology the traditional way.

Mizar was also used for teaching introductory logic courses at many universities in other countries, e.g., USA, Canada, Japan, Belgium.

Finally, Mizar has been used successfully as a formal environment to prepare lots of diploma theses, from the bachelor to PhD level.

2. Formalization of Mathematics Course

2.1 The Setting of the Course

During the spring semester of the 2003–2004 academic year, Mizar was used for the first time to conduct a course in "Formalization of Mathematics" (FM). The course was introduced as an obligatory part of the curriculum for all (57) first–year students of the Institute of Computer Science, University of Bialystok (UwB). The course consisted of 30 hours of lecture (given by A. Trybulec) and 30 hours of laboratory classes organized in 5 groups, each group lead by a different teacher (A. Trybulec, A. Naumowicz, A. Rybak, and the authors), with an average number of 11 students in each group.

The FM course followed a course called "Introduction to Logic and Set Theory", which the students had taken during the preceding semester. The course in logic and set theory was a short introduction referring to boolean operation on sets. It also employed Mizar as a tool to verify the correctness of reasoning. However its setting (30 hours of lecture, 15 hours of theoretical exercise classes, and 15 hours of laboratory classes) proved insufficient in making the students fluent users of the system. Still, the students attending the FM course were acquainted with basic Mizar proof constructions and selected proving techniques. Their knowledge was sufficient for understanding simple proofs as well as carrying out proofs of selected facts from set theory and relation theory by themselves.

2.2 Aims and the Domains of Mathematics Used in the Course

The main goals of the course were as follows:

– extending students' deduction skills by selecting exercises within the scope of the preselected areas of mathematics requiring the use of advanced proof techniques,

– teaching a selected mathematical theory presented during the lecture, and
– presenting the range of techniques used in formalization of mathematics.

Following the experience of past years, we have found that uninterpreted predicate calculus is not suitable for didactics. Students simply prefer working with interpreted theories. It is because only after mastering the essence of a given theory are the students able to solve a large number of tasks within one field of mathematics. At the same time, the theory chosen for the course should be rich enough to enable the presentation of all proof techniques.

Having in mind the above criteria and the university curriculum, we chose the theory of binary relations as the base of our FM course. The theory is sufficiently rich, even when the material for the exercises was notably restricted. The restriction was due to the realization of one of our main aims, namely, concentrating on teaching students various proof techniques which include:

– proofs by definitional expansion,
– conditional proof,
– proofs by "reductio ad absurdum" – a special kind of conditional proof in Mizar,
– proofs "per cases",
– proofs of existential statements.

To achieve this we used a set of specially prepared environments: RELATION, RELAT_AB and ENUMSET (the content of these environments is described in detail in 2.3).

The first environment was used to introduce relation theory, i.e., to present to students basic notions and definitions. It allowed for practicing several relation properties (e.g., symmetry, transitivity, reflexivity, etc.). The notions, definitions, and theorems included in two other environments were designed to extend students' knowledge on relation theory. They formed the basis for constructing proofs which use more complex techniques and proofs requiring creating counterexamples.

Apart from the relation theory, the course also included:

– set theory with the scheme of separation and
– natural number theory with the scheme of induction.

To enable working within these fields of mathematics, another two environments had been prepared (SEPARATE and INDUCT). However, they were not integrated with the previous ones. The above were included to the course program to meet requirements imposed by the university curriculum.

2.3 Some Technical Issues of the Course

The final didactic results of a computer–aided course like FM to a great extent rely on technical issues and earlier preparation. Being aware of this, a work meeting of the whole team involved in this experimental course was organized once a week. It allowed for anticipating selected problems and working out an effective teaching methodology. In many aspects these meetings gave the participants new insight into some aspects of teaching mathematics using Mizar.

The main question to cope with at the very beginning was whether the students should work on the whole MML repository or only on a part of it. In one discussion, it was decided that getting to work with the whole database would have been too troublesome for the students. In the result, it would have made using Mizar and fulfilling the aims quite complicated. Searching for suitable information in the whole database, even if the reference to source articles had been

given directly, could have appeared too complicated and time–consuming for the students. As a result, a decision was made to prepare special Mizar environments for all the notions presented during lectures, based on selected parts of MML. Consequently, five new environments were created:

- RELATION – unordered pairs of sets, relations, empty relation, inverse relations, various operations on relations (union, intersection, difference, symmetric difference, composition), inclusion, equality, and selected properties of relations (symmetry, asymmetry, antisymmetry, irreflexivity),
- RELAT_AB – domains and co-domains, images and inverse images of a set, relations on two sets, identity relations, Cartesian products, and selected properties of relations (reflexivity, weak connectedness, and connectedness),
- ENUMSET – singletons, ordered pairs, inclusion and separation of sets, operations on sets (union, intersection, difference, symmetric difference), and the empty set,
- SEPARATE – the scheme of separation, definitional theorems of the union and intersection of sets and the power set,
- INDUCT – the properties of the successor operator, the scheme of induction.

The first three environments were created successively on top of previous ones. The advantages of such an approach were the following:

- students could get familiar with more advanced mathematical notions step by step,
- teachers could prepare a comprehensive set of tasks for subsequent classes.

The second problem which appeared at the beginning of this didactic experiment was the proper choice of software for the students, i.e., a text editor to write their answers with and a version of the Mizar system to verify the results. GNU Emacs was chosen as the best editor for this purpose, since J. Urban's MizarMode for Emacs ([10]) (with a web–accessible manual) offered the students a complete user–friendly interface to the Mizar system. Because of the steady development of the Mizar system, it also required some thought about which of the system versions to use. Finally, the choice was made to employ version 7.0.02.

Another important decision was to use the Internet to facilitate the classes. Using the software prepared specially for the course by A. Naumowicz, students were able to download new environments for subsequent classes and store the results of their work after each class to individual accounts on a server. This technique allowed for successive control over students' progress and provided a good source for statistics.

During the course, three scored achievement tests were conducted. The scores contributed to each student's final grade as 30 – 30 – 40 per cent. The test routine was slightly more automatized than that of an ordinary class:

1. students had to log into their web–accounts, while a special program on the server generated an individual set of tasks to solve,
2. after downloading the tasks (and a new environment if needed) students tried to solve them on their computers,
3. after finishing their work, students uploaded their results on the server.

All students' submissions were later verified by the Mizar system and checked by the teacher. A task was assessed as solved correctly when the Mizar verifier reported no errors. The following scoring scheme was used: 0 points – any error appeared, 3 points (4 points in the last test) – no errors. The points collected by each students were used later to give a final grade from 2 (unsatisfactory) to 5 (very good). Mark statistics are given below in Table 1.

Table 1. Statistics of marks

Points	Marks	Number of students
0 - 25	2	7
26 - 30	3	11
31 - 35	3.5	7
36 - 40	4	19
41 - 45	4.5	8
46 - 50	5	5

2.4 Examples of Students Tasks

It seems that selecting the binary relation theory as a basis for the FM course was a good choice. Indeed, it allowed students to practice most of the proving techniques and solve a considerable number of tasks during the course.

The students' task was to prove from 1 to 5 theorems during each class (depending on the level of complexity). For 15 classes, about 140 theorems had been prepared in total. Most of them concerned the first three environments. However, not all of these theorems were proved by all students. In fact, an average student proved about 55 theorems.

To illustrate the level of complexity, we recall some examples of theorems which students had to prove during the course (Table 2). We also present two examples of theorems with proofs in Table 3.

3. Conclusions

The data collected during the course allows us to draw some conclusions about the effects of our teaching, and to analyze the biggest sources of problems and ways of eliminating them for planning similar course. In our minds, the overall didactic result of this experiment was positive. The average grade was close to 4 (on a scale of 2 – 5). Undoubtedly, this effect was caused by the following factors:

- individual students' work was under the control of the teacher,
- a comprehensive set of relatively easy tasks allowed students to get some satisfaction from solving several exercises, and at the same time gave them a chance to acquire the mathematical content of the lectures,
- the ability of individuals to work at home (controlled by the Mizar system) enabled more intensive training in the art of proving.

Apparently, the biggest problems in students' work concerned the errors reported by Mizar in their proofs. Students did not pay enough attention to the comments flagged by the system. Some students seemed to not understand them at all or very often misinterpreted them. Even worse, they did not report these problems to the teacher which significantly slowed down the process of proving in class. Therefore, it seems that to minimize the problem the sets of tasks could be enriched with exercises based on individual recognizing and improving errors in specially prepared erroneous proof texts.

Another set of exercises of particular interest should be based on constructing only the skeletons of proofs (correct solutions can contain only errors 4 or 1). This is because efficient proving absolutely requires the knowledge of what proof strategy can be used in a given context, what assumptions can be made, and, finally, what the current goal of reasoning (thesis) is. Many of the students involved in our course did not have this skill developed enough.

Table 2. Some examples of the students' tasks

Examples	

property of relation	`for P,R being Relation st P c= R & R is irreflexive`
	`holds P is irreflexive;`
	
	`for P being Relation holds`
	`P is transitive implies P~ is transitive;`
	
	`for X being set, R being Relation of X,X`
	`st R is connected holds R is weakly_connected &`
	`R is reflexive;`

"per cases"	`for R being Relation holds`
	`dom R = {} or rng R = {} implies R = {};`
	
	`for R,S being Relation holds R~ \/ S~ c= (R \/ S)~;`

counterexamples	`ex R,S being Relation st R is asymmetric &`
	`S is asymmetric & not R*S is asymmetric;`
	
	`ex R,S being Relation st R is irreflexive &`
	`S is irreflexive & not R*S is irreflexive;`

scheme of separation	`for x being set, ex A being set`
	`st for y being set holds y in A iff y = x;`
	
	`for A,B being set ex C being set`
	`st for x being set holds x in C iff x in A & x in B;`

scheme of induction	`for i,j,k being natural number`
	`holds i*(j+k) = i*j + i*k;`
	
	`for i,j,k being natural number`
	`holds i+k = j+k implies i = j;`

Other conclusions and solutions:

1. Modifying, to a certain extent, the program content or adding new elements (paying special attention to the parts of the material which, when acquired to an unsatisfactory degree, result in students' ineffective work during classes).
2. Changing the order of introducing new environments in subsequent classes, for example, perhaps starting in the two last and going to binary relation theory after that.
3. At the very beginning obliging students to get familiar with
   – the chosen Mizar syntax,
   – the editor used during classes.
4. Introducing automatically generated sets of exercises for each class to make students' work more independent and disable cheating in class by "rewriting" neighbour's work.
5. Motivating students to do home work assignments and working out ways to assess them.

4. Plans and Future Work

The university curriculum planned for the next year contains two computer–aided subjects based on Mizar: "Introduction to Mathematics" and "Formalization of Mathematics". Both these courses will be obligatory for first–year computer science students at UwB, the former conducted

Table 3. Examples of theorems with proofs

```
reserve P,R,S,T for Relation;
reserve x,y,z for set;

P is symmetric & R is symmetric implies P\/R is symmetric
  proof
    assume
A1: P is symmetric & R is symmetric;
    let x,y;
    assume [x,y] in P\/R;
    then [x,y] in P or [x,y] in R by RELATION:4;
    then [y,x] in P or [y,x] in R by A1,RELATION:def 12;
    hence thesis by RELATION:4;
  end;
```

```
ex R,S,T st R*(S/\T)<>(R*S)/\(R*T)
  proof
    reconsider R={[0,1],[0,2]} as Relation by RELATION:9;
    reconsider S={[1,3]}, T={[2,3]} as Relation by RELATION:8;
    take R,S,T;
A1: S/\T={}
    proof
      assume S/\T<>{};
      then consider x,y such that
B1:   [x,y] in (S/\T) by RELATION:def 3;
      [x,y] in S & [x,y] in T by B1,RELATION:5;
      then [x,y]=[1,3] & [x,y]=[2,3] by ENUMSET:def 1;
      hence contradiction by ENUMSET:2;
    end;
A2: R*(S/\T)= {} by RELATION:10, A1;
A3: [0,1] in R by ENUMSET:def 2;
    [1,3] in S by ENUMSET:def 1; then
A4: [0,3] in R*S by RELATION:3,A3;
A5: [0,2] in R by ENUMSET:def 2;
    [2,3] in T by ENUMSET:def 1;
    then [0,3] in R*T by RELATION:3,A5;
    then [0,3] in (R*S)/\(R*T) by A4,RELATION:5;
    hence R*(S/\T)<> (R*S)/\(R*T) by A2,RELATION: 1;
  end;
```

in the autumn semester, the latter in the spring semester. The “Introduction to Mathematics” course will also be run in parallel for all first–year mathematics students. The time setting of each course is as follow: 30 hours of lecture + 30 hours of laboratory classes. According to this new time arrangement, the content of the first course will include:

- all the material from both semesters of the previous academic year, and
- extra tasks which we think should be included in the program of the course, but were not completely realized the previous year.

We plan to achieve this by a proper choice of exercises and improving class organization, and also students’ individual work based on the experience they get in class. This change, we believe, should improve the effectiveness of students’ work.

Some parts of the mathematical content of classes will be revised. We would like to include also more advanced fields of mathematics together with more complicated mathematical constructs, e.g., structures.

In the long run, we intend to create an integral CAI approach for university students. Our idea is based on distinguishing three stages of instruction, each stage with a different level:

- Freshmen level. On this level we propose teaching the basics of the system, which will be used and teaching – how to prove. The materials used in this level should be limited by the study programme.
- Intermediate level. This level refers to course-ware for selected fields of mathematics. The teacher should be able to select parts of mathematics which are best for this educational process.
- Research level. Tools of computer based e–learning mathematics should be applied to document students' own work up to the level of PhD thesis.

We expect to obtain in this way a systematic approach to computer–aided instruction employing the Mizar system (the system used for teaching purposes will be further used, e.g., for doing research). We believe that the use of the system will actually facilitate the process of teaching mathematics. But, still the open problem is how many courses should be processed in this way.

However we hope that **mathematics practice will become more egalitarian** ([2]) for our students.

Acknowledgments

We would like to thank all those who helped us with writing this paper for their valuable hints and advice. In particular we want to thank A. Trybulec, A. Naumowicz, and P. Rudnicki.

References

1. Bajguz W., Czuba S.T., *Zbiór zadań z topologii w PC Mizar (A Collection of Exercises on Topology in PC Mizar)*, Zeszyt 1–5, Oddzial Regionalny Ogólnopolskiej Edukacji Komputerowej, Bialystok, 1989–90
2. Farmer M.W., *Open Challenges of Computerized Mathematics* Mc–Master University Hamilton, Ontario, Canada, September 12, 2003 [slides from CALCULEMUS–I, Rome 2003]
3. Jaśkowski S., *On the Rules of Supposition in Formal Logic*, Studia Logica I, 1934, Warszawa *Reprinted in Polish Logic*, ed. S.McCall, Clarendon Press, Oxford 1967
4. Kordos M. and Szczerba L., *Geometria dla nauczycieli (Geometry for Teachers)*, PWN, Warsaw, 1976
5. Mostowski M. and Trybulec Z., *A Certain Experimental Computer Aided Course of Logic in Poland*, Proceedings of World Conference on Computer in Education, IFIP/AFIPS, Norfolk, North Holland, 1985
6. Prażmowski K. and Rudnicki P., *Kurs Logiki w Mizarze–MSE (Course in Login in Mizar–MSE)*, Monthly DELTA, No 9 and next ones, Warsaw, 1983–1984
7. Szczerba L.W., *The Use of Mizar–MSE in a Course in Foundations of Geometry*, Initiatives in Logic, Nijhoff Publishers, Dordrecht, 1987
8. Trybulec A., *Logic Information Language MIZAR–MSE* Institute of Computer Science Polish Academy of Sciences, No. 465 Warsaw, 1982
9. Zalewska A., *An Application of Mizar–MSE in a Course in Logic*, Initiatives in Logic, Nijhoff Publishers, Dordrecht, 1987
10. Urban J., *Mizar Mode for Emacs*, **http://alioth.uwb.edu.pl/twiki/bin/view/Mizar/MizarMode**

MECHANIZED MATHEMATICS AND ITS APPLICATIONS, VOL. 4, NO. 1, MARCH 2005, 43:50

Computers and Algorithms in the Mizar System

Artur Korniłowicz[1]* and Christoph Schwarzweller[2]**

[1] Institute of Computer Science, University of Białystok, Białystok, Poland **artur@math.uwb.edu.pl**
[2] Department of Computer Science, University of Gdańsk, Gdańsk, Poland **schwarzw@math.univ.gda.pl**

Abstract – We review the general model of a computer as defined in the Mizar system. The main emphasis is on specializing and using this model to formulate algorithms and prove their correctness. We give examples for a concrete machine over the integers derived from the general model. Further development of this model – especially towards the complexity of algorithms – is discussed.

1. Introduction

The development of the Mizar system [Miz04] started 30 years ago with the proposition of a formal language intended to match mathematical vernacular. The initial goal of this language – also called Mizar [AT78] – was not to write or check mathematical proofs but rather to assist mathematicians in writing and preparing their papers. Since then, however, Mizar has evolved into a proof checker [WWW] with the largest known repository of formally verified mathematical knowledge covering a number of topics such as analysis, algebra, topology, graph theory, category theory, and others. The most ambiguous projects have been the formalization of a text book on continuous lattices [CCL80] and the formalization of Jordan's curve theorem [CJ87].

Not widely known outside the Mizar community, however, are the efforts concerning programs and program verification in Mizar. Beginning in the early 90's of the last century a theory of abstract computers has been developed and formalized in Mizar. The basic Mizar model of a computer [NT92] is quite general giving a conceptual framework in which computations take place. On top of this a number of so-called macro instructions resembling the structure of familiar procedural programming languages has been introduced and used to show the correctness of several algorithms, e.g., sorting algorithms.

In this paper we give an overview of abstract computers and their algorithms as defined in Mizar. Based on the general abstract computer, we discuss the possibilities of specializing this model into different – concrete and generic – machines. After that we concentrate on the **SCM-FSA** machine [TNR96], a "Simple Concrete Model" of a computer over the integers. We present the mechanism of macro instructions which makes it possible to formulate algorithms for this machine in a programming language fashion. Finally we consider the question of how to describe and prove the complexity of algorithms. The basics of complexity theory have been formalized in Mizar [KRS01b], however they have not yet been applied to algorithms for abstract computers.

2. The Mathematical Model of a CPU

The basic model of a computer has been defined in a quite general way [NT92]. It does not resemble a special kind of programming language nor even a particular kind of machine. The

* The paper was written during the first author's post-doctoral scholarship granted by Shinshu University, Japan.
** The work of the second author was partially supported by grant BW 5100-5-0147-4.

Manuscript received August 6, 2004; revised September 10, 2004

idea was to capture the general structure of computations on a machine in this way to provide a framework which can be instantiated to realize different models of computation. Consequently, the Mizar computer in fact models a central processing unit by introducing among others the concept of instructions and an instruction counter. This means that memory and memory management is explicitly addressed. Here is the definition of the structure **AMI-Struct** [NT92].

```
definition let N be set;
struct (1-sorted) AMI-Struct over N
  (# carrier -> set,
     Instruction-Counter -> Element of the carrier,
     Instruction-Locations -> Subset of the carrier,
     Instruction-Codes -> non empty set,
     Instructions -> non empty Subset of
                   [:the Instruction-Codes,((union N) \/ the carrier)*:],
     Object-Kind -> Function of the carrier,
                       N \/ {the Instructions,the Instruction-Locations},
     Execution -> Function of the Instructions,
              Funcs(product the Object-Kind, product the Object-Kind) #);
end;
```

The **carrier** plays the role of memory and the set **N** gives the data that can be stored within it. Note that both the instruction counter and the locations of instructions are placed in the same memory. The functor **Object-Kind** indicates which kind of information – data or instructions – can be stored in a given memory location. The state of a computer is thus a function from the memory – the **carrier** – which respects these limitations and is hence an element of **product Object-Kind**. Finally, an instruction is given by an instruction code and a data or memory element. Thus, for example [J,<*a*>] can be read as the instruction "jump to a", i.e., the instruction counter is set to a. The interpretation of an instruction, however, is not fixed. Its meaning is given by the functor **Execution** which maps each instruction to a function from states to states.

This definition just establishes the basic concepts of a central processing unit and thus does not give a machine with the usual expected properties. Such properties – if desired – are enforced in a second step using Mizar attributes. A natural restriction, for example, is that only instruction locations can be put into the instruction counter. As a second example, note that in principle the execution of an instruction can change given program by overwriting the value of an instruction location. A machine, in which this does not occur, is called *steady programmed*.

Note also that the memory is an unordered set, which means in particular that it is not possible to speak of the k-th instruction (location) of a machine. To overcome this, an ordering of instruction locations has been defined in [TRK01]. The ordering is given by a sequence of instruction locations that can be put into the instruction counter during its execution. A machine in which there exists a bijection between the natural numbers and the instruction locations respecting this order is called a *standard* machine.

In addition, parameterization by the set **N** makes it possible to consider special purpose machines: **N** gives the data that can be stored, so one could for example define a machine storing integer numbers or finite sequences of integers (the **SCM** and **SCM-FSA** computer respectively, see [NT93,TNR96]). Note that by instantiating the parameter **N** operations of the data also become available, in our example addition of integer numbers or concatenating finite sequences. These operations can be considered as the primitives of the machine and are used when defining the functor **Execution**, which gives the effect of the instructions. It is even possible to define generic machines that work over arbitrary algebraic structures: **N** is instantiated with the (algebraic) structure – realized in Mizar by structure definitions. Then, the elements of the structure become the data and the operations of the structure become the primitives of the

machine. For example an **SCM**-Ring machine working over arbitrary rings has been defined in [Kor98].

Programming such a machine is not an easy task, it is basically the same as programming in an assembler language. An obvious goal then is to extend the model by introducing higher-level programming constructs that resemble ordinary programming languages. This has been done for the **SCM-FSA**-machine [TNR96], an extension of the **SCM**-computer over the integers, which in addition allows for the storing of finite sequences of integers.

SCM, **SCM-FSA** and **SCM**-Ring are standard in the sense mentioned above. But, obviously, not all machines that can be defined based on `AMI-Struct` must be standard. An example is the "Small Computer Model with Push-Down Stack" (**SCMPDS**) defined in [Chen99b,Chen99c]. Here shifting by a fixed number of locations forward or backward is allowed.

3. Macros

To facilitate the writing of programs and algorithms and making them more readable (more similar to programs written in languages like Pascal, C, etc.) a number of macros have been introduced in MML. They allow for the writing of programs in terms of Mizar terms. In this section we present two different definitions of macros and discuss the impact of these definitions on how macros are composed.

3.1 The General Definition

As a very basic definition of a macro one can take the finite partial state of a computer such that the domain of it is a subset of instruction locations. In fact, it has been done in this way in MML, but one more condition has been added. Because it was done for **SCM-FSA** [TNA97], and because the instruction locations of **SCM-FSA** are constituted by natural odd numbers the authors decided to assume that a macro must be *initial*, meaning that a macro must occupy the contiguous part of the memory starting from the first location. Having such a condition one can assume that, when a macro is executed, we can (must) always start the execution from the first instruction location. We do not put any conditions on the structure of the macro. Then, the composition of two macros (let's say f and g) can be done in the following steps [TN96a,TN96b]:

- incrementation of all jumps of g by the length of f,
- shift of incremented g by the length of f,
- change of all halts of f by jumps to the first location of incremented and shifted g,
- concatenation of new f with new g.

3.2 A More Specific Definition

From some points of view, the previous definition of a macro is not convenient. Why? When one wants to combine two macros, one must replace all halts of the first macro by jumps to the beginning location of the second one. Therefore it was decided to introduce a new definition of a macro, which is more suitable for concatenation. Namely, authors added the condition that a macro must be finished by *halt* instruction, and only one *halt* instruction is allowed, in contrast to the previous definition [TRK01]. Of course, this definition is more restrictive than the original one, but to compose two macros it is enough to remove the last instruction of the first macro and concatenate the second one, instead of changing all halts by jumps. Then, the composition of two macros (let's say f and g) can be done in the following steps [Kor01]:

– incrementation of all jumps of g by the length of f minus 1,
– shift of incremented g by the length of f minus 1,
– remove of the last instruction of f,
– concatenation of new f with new g.

The shortness of the second method of composition is clear in the case when one concatenates macro instructions made of one “non halt” and one “halt” instruction. Table 1 presents the composition of four such macro instructions. Following the first method of concatenation we obtain a macroinstruction containing 3 superfluous jumps as a result. They do not appear in the case of the second definition. (In Table 1, the notion il.n stands for the n-th instruction location, and dl.n for the n-th data location.)

Table 1. An example of the concatenation of macro instructions

					general definition	specific definition
il.n	M_1	M_2	M_3	M_4	$M_1;M_2;M_3;M_4$	$M_1;M_2;M_3;M_4$
il.0	dl.1:=dl.0	dl.2:=dl.0	dl.3:=dl.0	dl.4:=dl.0	dl.1:=dl.0	dl.1:=dl.0
il.1	**halt**	**halt**	**halt**	**halt**	**goto** il.2	dl.2:=dl.0
il.2					dl.2:=dl.0	dl.3:=dl.0
il.3					**goto** il.4	dl.4:=dl.0
il.4					dl.3:=dl.0	**halt**
il.5					**goto** il.6	
il.6					dl.4:=dl.0	
il.7					**halt**	

3.3 Examples of Macros

There is a number of macros already introduced in MML. We present macros defined only for **SCM-FSA** [Asa97a,Asa97b,Asa97c]. There are macros in the sense of the general definition, but not always in the sense of the second definition (not all of them end by *halt*).

1. Unconditional jump:

```
definition
 let l be Instruction-Location of SCM+FSA;
 func Goto l -> Macro-Instruction equals :: SCMFSA8A:def 2
  insloc 0 .--> goto l;
end;
```

2. Conditional macro:

```
definition
 let a be Int-Location;
 let I, J be Macro-Instruction;
 func if=0(a,I,J) -> Macro-Instruction equals :: SCMFSA8B:def 1
 a =0_goto insloc (card J + 3) ';' J ';' Goto insloc (card I + 1) ';'
     I ';' SCM+FSA-Stop;
end;
```

3. The below repeats a times I :

```
definition
 let a be Int-Location;
 let I be Macro-Instruction;
 func Times(a,I) -> Macro-Instruction equals :: SCMFSA8C:def 5
  if>0(a,loop if=0(a,Goto insloc 2,I ';' SubFrom(a,intloc 0)),
     SCM+FSA-Stop);
end;
```

4. Algorithms in Mizar

Development of the theory of computers, especially the theory of macro instructions in MML, started and it is being still continued with deep confidence that such computers will be (are) suitable for the verification of algorithms. A typical way of saying that an algorithm is correct relies on the comparison of the algorithm with a function that describes the semantics of the algorithm. The comparison should be done in terms of the predicate **computes** [NT92], defined as:

```
definition
  let N be with_non-empty_elements set;
  let S be realistic halting IC-Ins-separated definite
        (non empty non void AMI-Struct over N);
  let p be FinPartState of S, F be Function;
  pred p computes F means :: AMI_1:def 29
  for x being set st x in dom F ex s being FinPartState of S st x = s &
   p +* s is pre-program of S & F.s c= Result(p +* s);
 end;
```

The function mentioned above is usually a partial function from the sets of all finite partial states to the sets of all finite partial states of the computer such that elements of the domain are related to inputs to the algorithm and elements of the codomain to results of the algorithm. What is important is that it is not necessary to define a function for each algorithm. If different algorithms solve exactly the same problem, it means that they compute the same function.

As an example let us take into account the Euclidean algorithm defined and verified in [TN93]. The algorithm is just the assignment of appropriate instructions to the consecutive instruction locations.

```
definition
func Euclide-Algorithm -> programmed FinPartState of SCM
   equals :: AMI_4:def 1
   (il.0 .--> (dl.2 := dl.1)) +*
  ((il.1 .--> Divide(dl.0,dl.1)) +*
  ((il.2 .--> (dl.0 := dl.2)) +*
  ((il.3 .--> (dl.1 >0_goto il.0)) +*
   (il.4 .--> halt SCM))));
end;
```

The corresponding function can be defined as:

```
definition
 func Euclide-Function -> PartFunc of FinPartSt SCM, FinPartSt SCM
    means :: AMI_4:def 2
  for p, q being FinPartState of SCM holds [p,q] in it
   iff ex x, y being Integer st x > 0 & y > 0 &
    p = (dl.0,dl.1) --> (x,y) & q = dl.0 .--> (x gcd y);
end;
```

It assigns partial states `q` of **SCM** that store the result (`x gcd y`) to relevant partial states `p` containing inputs (`x,y`).

A more advanced algorithm that has already been defined and verified using Mizar is bubble sorting of a finite sequence of numbers [CN98]. It involves macros in its body and looks like:

```
definition
  set a0 = intloc 0, a1 = intloc 1, a2 = intloc 2, a3 = intloc 3;
  set a4 = intloc 4, a5 = intloc 5, a6 = intloc 6;
  set initializeWorkMem =
      (a2:=a0) ';' (a3:=a0) ';' (a4:=a0) ';' (a5:=a0) ';' (a6:=a0);
 let f be FinSeq-Location;
 func bubble-sort f -> Macro-Instruction equals :: SCMBSORT:def 1
   initializeWorkMem ';' (a1:=len f) ';'
   Times(a1, (a2:=a1) ';' SubFrom(a2,a0) ';' (a3:=len f) ';'
        Times(a2, (a4:=a3) ';' SubFrom(a3,a0) ';'
             (a5:=(f,a3)) ';' (a6:=(f,a4)) ';' SubFrom(a6,a5) ';'
             if>0(a6, (a6:=(f,a4)) ';' ((f,a3):=a6) ';' ((f,a4):=a5),
                      SCM+FSA-Stop)));
end;
```

To verify its correctness the following function has been introduced. This function assigns partial states `q` containing sorted sequences `u` to partial states `p` containing original sequences `t`.

```
definition
 func Sorting-Function -> PartFunc of FinPartSt SCM+FSA,FinPartSt SCM+FSA
     means :: SCMBSORT:def 3
  for p, q being FinPartState of SCM+FSA holds [p,q] in it
   iff ex t being FinSequence of INT, u being FinSequence of REAL
    st t,u are_fiberwise_equipotent & u is FinSequence of INT &
       u is non-increasing & p = fsloc 0 .--> t & q = fsloc 0 .--> u;
end;
```

We close this section with a list of other algorithms included in MML:

- **SCM**
  - Fibonacci numbers – FIB_FUSC:def 1 ([BR93b])
  - computing of nominators and denominators of all rationals – FIB_FUSC:def 3 ([BR93b])
- **SCM-FSA**
  - insertion sort algorithm – SCMISORT:def 2 ([Chen99a])
- **SCMPDS**
  - quick sort algorithm – SCPQSORT:def 2 ([Chen01c])
  - insertion sort algorithm – SCPISORT:def 2 ([Chen01b])
  - greatest common divisor using subtraction – SCPINVAR:def 4 ([Chen01d])
  - greatest common divisor using recursion – SCMP_GCD:def 4 ([Chen01a])
  - Fibonacci sequence – SCPINVAR:def 2 ([Chen01d])

5. Outlook: Complexity of Algorithms

Formalizing the complexity of algorithms on a machine is a complicated task. One needs a formalization of both the theoretical basis of complexity and a notion that connects formalized

algorithms with this basis. The usual approach is to define step counting functions where often not all but only the steps of main interest – e.g., multiplications or comparisons – are considered.

The beginnings of complexity theory has already been formalized in Mizar [KRS01a,KRS01b]. Here the basic notations such as domination and complexity classes are introduced. In addition, a number of examples and problems from a textbook have been addressed. Thus [KRS01a,KRS01b] give a framework for defining the complexity of algorithms. This framework, however, has not yet been applied to the description of the complexity of algorithms. In the following, we briefly outline how this can be done.

Mizar computers provide a number of possible "one-step" operations on states, from which programs, that is algorithms, are built. It is therefore natural to consider sequences of execution steps to capture the length of a computation. In [NT92] the concept of a computation sequence – to identify halting computations – has already been introduced. It was defined using a functor **`Following`** computing the immediate successor of a state **`s`**. Thus **`Computation s`** is the sequence of states the machine is running through when initialized with state **`s`**. Using this functor it is now straightforward to define a step function, that counts the number of executions of the machine beginning with a state **`s`** [BR93a]. **`CurInstr`** gives the instruction of a state, i,.e., the instruction to be executed next.

```
definition
 let N be with_non-empty_elements set;
 let S be halting IC-Ins-separated definite
                        (non empty non void AMI-Struct over N);
 let s be State of S such that s is halting;
 func Complexity s -> Nat means :: SCM_1:def 2
  CurInstr((Computation s).it) = halt S &
  for k being Nat
         st CurInstr((Computation s).k) = halt S holds it <= k;
 end;
```

Note that a computation is in principle infinite. Therefore, the attribute **`halting`** describes an instruction that does not change a state is used to indicate the "end" of a computation. Using this definition it has been proved, for example, that the **SCM**-program computing Fibonacci numbers mentioned in the last section makes $6 * \mathbf{N} - 2$ steps (e.g., **`Complexity s = 6 * N - 2`**) if the input **`N`** is greater than 0.

The functor can be easily modified to count only the execution steps that involve primitive operations given by the parameter **`N`**. This means basically summing up the elements in **`Computation s`** with a certain instruction code. Counting functions for only one operation such as multiplication or comparison can be constructed analogously. Thus based on the general machine model defined in Mizar, it is possible to define step-counting and complexity functions following the literature.

6. Conclusion

We have presented the general computing machine defined in Mizar. The macro mechanism makes it possible to use this machine to formulate algorithms and prove them correct by showing which function the algorithm computes. In principle, this approach works for arbitrary programs though it seems hard to apply it to algorithms occurring in everyday life. We believe, however, that this approach is suitable for formalizing the principles of algorithms – as presented in textbooks on this topic for example – in order to store them in a mathematical repository.

The development of a complexity theory for algorithms as mentioned above is still in its beginning phases. The complexity function presented in [BR93a], for example, only considers

a single state **s**. To describe the complexity of algorithms it should be generalized to a function from all states of a machine **s** – or some kind of input for **s**.[3] To summarize, the approach formalized so far has still to be extended and worked out in order to become suitable for proving the complexity of algorithms.

References

[Asa97a] N. Asamoto, Conditional branch macro instructions of SCM+FSA, Part I (preliminary); Formalized Mathematics, 6(1), pp. 65–72, 1997.

[Asa97b] N. Asamoto, Conditional branch macro instructions of SCM+FSA, Part II; Formalized Mathematics, 6(1), pp. 73–80, 1997.

[Asa97c] N. Asamoto, The loop and times macroinstruction for SCM+FSA; Formalized Mathematics, 6(4), pp. 483–497, 1997.

[AT78] A. Trybulec, The Mizar-QC/6000 Logic Information Language, ALLC Bulletin, Vol.6, No 2, 1978.

[BR93a] G. Bancerek and P. Rudnicki, Development of Terminology for SCM; Formalized Mathematics, 4(1), pp. 61–67, 1993.

[BR93b] G. Bancerek and P. Rudnicki, Two Programs for SCM. Part II - Programs; Formalized Mathematics, 4(1), pp. 73–75, 1993.

[CCL80] G. Gierz, K.H. Hofmann, K. Keimel and J.D. Lawson, M. Mislove and D.S. Scott, A Compendium of Continuous Lattices, Springer-Verlag, 1980.

[Chen99a] J. Chen, Insert Sort on SCMFSA; Formalized Mathematics, 8(1), pp. 119–127, 1999.

[Chen99b] J. Chen, A Small Computer Model with Push-Down Stack; Formalized Mathematics, 8(1), pp. 175–182, 1999.

[Chen99c] J. Chen, The SCMPDS Computer and the Basic Semantics of its Instructions; Formalized Mathematics, 8(1), pp. 183–191, 1999.

[Chen01a] J. Chen, Recursive Euclide Algorithm; Formalized Mathematics, 9(1), pp. 1–4, 2001.

[Chen01b] J. Chen, Insert Sort on SCMPDS; Formalized Mathematics, 9(2), pp. 407–412, 2001.

[Chen01c] J. Chen, Quick Sort on SCMPDS; Formalized Mathematics, 9(2), pp. 413–418, 2001.

[Chen01d] J. Chen, Justifying the Correctness of the Fibonacci Sequence and the Euclide Algorithm by Loop-Invariant; Formalized Mathematics, 9(2), pp. 419–427, 2001.

[CJ87] C. Jordan, Cours d'Analyse de l'École Polytechnique, 1887.

[CN98] J. Chen and Y. Nakamura, Bubble Sort on SCM+FSA; Formalized Mathematics, 7(1), pp. 153–161, 1998.

[Kor98] A. Korniłowicz, The Construction of SCM over Ring; Formalized Mathematics, 7(2), pp. 295–300, 1998.

[Kor01] A. Korniłowicz, On the Composition of Macro Instructions of Standard Computers; Formalized Mathematics, 9(2), pp. 303–316, 2001.

[KRS01a] R. Krueger, P. Rudnicki and P. Shelley, Asymptotic notation. Part I: Theory; Formalized Mathematics, 9(1), pp. 135–142, 2001.

[KRS01b] R. Krueger, P. Rudnicki and P. Shelley, Asymptotic notation. Part II: Examples and Problems; Formalized Mathematics, 9(1), pp. 143–154, 2001.

[Miz04] The Mizar Home Page, **http://mizar.org**.

[NT92] Y. Nakamura and A. Trybulec, A Mathematical Model of CPU; Formalized Mathematics, 3(2), pp. 151–160, 1992.

[NT93] Y. Nakamura and A. Trybulec, Some Remarks on Simple Concrete Model of Computer; Formalized Mathematics, 4(1), pp. 51–56, 1993.

[RT01] P. Rudnicki and A. Trybulec, Mathematical Knowledge Management in Mizar; in: B. Buchberger, O. Caprotti (eds.), Proceedings of the First International Workshop on Mathematical Knowledge Management (MKM2001), Linz, Austria, 2001.

[TN96a] A. Trybulec and Y. Nakamura, Modifying addresses of instructions of SCM+FSA; Formalized Mathematics, 5(4), pp. 571–576, 1996.

[TN96b] A. Trybulec and Y. Nakamura, Relocability for SCM+FSA; Formalized Mathematics, 5(4), pp. 583–586, 1996.

[TNA97] A. Trybulec, Y. Nakamura and N. Asamoto, On the compositions of macro instructions; Formalized Mathematics, 6(1), pp. 21–27, 1997.

[TN93] A. Trybulec and Y. Nakamura, Euclid Algorithm; Formalized Mathematics, 4(1), pp. 57–60, 1993.

[TNR96] A. Trybulec, Y. Nakamura, and P. Rudnicki, The SCM+FSA computer; Formalized Mathematics, 5(4), pp. 519–528, 1996.

[TRK01] A. Trybulec, P. Rudnicki, and A. Korniłowicz, Standard Ordering of Instruction Locations; Formalized Mathematics, 9(2), pp. 291–301, 2001.

[3] This was pointed out by one of the referees.

MECHANIZED MATHEMATICS AND ITS APPLICATIONS, VOL. 4, NO. 1, MARCH 2005, 51:58

Robustness of Systems for Formalizing Mathematics – Testing Monotonicity and Permutability of References in MIZAR

Robert Milewski

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

Abstract – When thinking of various systems for formalizing mathematics and databases associated with them, one may question whether a given system is "robust" and what criteria may enable us to make such a statement. This issue may be considered quite differently depending on the system being examined. The work presented here concerns the MIZAR system.
Statements in MIZAR can be justified by providing a list of references yielding their correctness. This paper describes some experiments testing the robustness of the MIZAR system which consist of modifying lists of references (through altering their order – testing permutability, or extending them with extra references – testing monotonicity).
After implementing suitable programs and performing numerous experiments it turned out that there were cases in which the system did not fulfill the principle of permutability and monotonicity of references. Initially, cases in which the justification of a statement was not accepted by the system after the order of references was changed were observed in four MML articles. In another two articles there were cases in which the system did not accept statements when a redundant reference had been added. Consequently, these situations were analysed and, after applying suitable patches, the problems were eliminated.
Repeating the same tests with updated versions of the system made it possible to find another problem connected with the monotonicity of references. The results of these experiments allow us to claim that it is necessary to constantly check the system by repeating the described tests in order to improve its robustness.

1. Motivation and Main Ideas

MIZAR [1], [7] is a system for computer-aided formalization of mathematics created by Andrzej Trybulec. The beginning of this system dates back to the 1970s, but since that time the system has been constantly modified, improved, and strengthened.

In parallel to the development of the system, a database of computer-checked mathematical knowledge, *Mizar Mathematical Library* (MML), has been developed since 1989. As with the Mizar system, MML is also subject to constant reorganization, mainly aimed at making better use of the information it contains.

The multitude of systems for formalizing mathematics forces us to compare and classify them. Comparisons of this kind have already been done for some time taking into account various factors. These can be the size of database, the power of logic, or the level of automation offered by given systems among others. One such attempt is presented in Freek Wiedijk's work [15] showing proofs presented in different languages of the fact that the square root of 2 is irrational.

The experience of researchers working with systems for formalizing mathematics indicates that we need to think about the robustness of particular systems, i.e., to what extent they are powerful and reliable in different untypical situations. Such situations may often occur during normal usage.

Manuscript received August 7, 2004; revised September 2, 2004

The object of this study is the MIZAR system ([11], [12]), which I have been using for many years now. Over that period of time, I have encountered various untypical situations which pushed me to a deeper analysis of this issue, which I generally call "studying robustness".

One of the methods to justify the correctness of a statement in MIZAR is by providing a list of references that should yield the statement. The order of references in such a list is not strictly fixed, it depends on the user (there are programs altering the order according to given criteria, e.g., a tool for sorting them alphabetically). In MIZAR, the order of references should not be relevant to the acceptance of a sentence being proved. Similarly, adding references not related to the justification of a sentence should not be significant. I call the above issues "testing permutability of references" and "testing monotonicity of references", respectively.

The need for the study of these issues becomes even more important as the authors of articles do not always pay attention to untypical situations they find while working with the system and do not always inform the authors of the system. Very often their goal is simply to justify subsequent reasoning steps and even in situations where they observe that the system works incorrectly, they try to solve their problem locally without notifying the system's developers.

2. References in MIZAR

There are a couple of methods of justifying statements in MIZAR. In the simplest case, statements are accepted by the system automatically as a result of applying rules of propositional calculus, built-in information taken from requirements imported by a given article, or properties of definitions assigned to the used predicate or functor [9].

The second method of justifying the correctness of a sentence is obtained by providing a formal proof of the given fact.

In the third case, if we can justify the statement in one step by referring to other facts, we use the straightforward justification method. This means that after the statement we put the keyword "**by**" and then list the labels of sentences to which we want to refer – statements from the current article as well as external theorems. Additionally we may refer to the previous statement putting the word "**then**" before the justified statement:

```
A1: sentence1;
A2: sentence2;
sentence3;
then sentence4 by A1,A2,XBOOLE_0:def 1;
```

In the above example, we refer to **sentence1** and **sentence2** through the labels **A1** and **A2**, respectively, and to **sentence3** through the use of **then** and the definition of the empty set **XBOOLE_0:def 1**. When we want to state the conclusion while referring to a previous statement we use the word "**hence**" (the combination "**then thus**" is not allowed).

There are no rules imposed on the order of references in a straightforward justification, it is up to the authors of the article. There is a utility program available which can be used to sort them lexicographically. In MIZAR, the order of references should not matter for the acceptance of a statement by the system. However, some users reported situations where it seemed to be important, which proved that MIZAR was not free of such faults, hence not fully robust. This confirmed the need to create software to test the entire MML in order to find similar problematic cases.

Similar is the case of adding extra (redundant) references. Statements which are accepted with a given list of references should also be accepted with an extended list, as it should not be

relevant to the *Verifier*. Apparently, as in the case with permutability, users reported some facts which contradicted this postulate.

3. Preparing MML for the Experiments with Delinker

To carry out experiments on the permutability of references the articles from MML had to be specially prepared. The first problem to occur regarded linking to previous statements using the keyword **then** or concluding with the word **hence**. A reference introduced by **then** or **hence** is treated as the first reference in an inference step, but it cannot be simply moved simply to other positions, so it cannot be permuted. Solving the problem required implementing a program to replace all occurrences of **then** with references to the previous sentence and put in the list as a first reference so as to not change the original order. In the case of linking with **hence**, the keyword had to be replaced with the word **thus**, to preserve the conclusion. Sometimes the previous statement had to be extra labeled, when there was no label in the original text. To prevent conflicts, i.e., duplicating labels, it seemed the best solution to change all label identifiers according to one naming scheme. A program, which I call Delinker (DELLINK) inserts labels in every possible location and also changes the names of existing ones in the following way: the names of labels are of the form Rx:, where the values of x are consecutive numbers, starting from 1. Delinker also removes linking with **then** and changes **hence** into **thus**, adding to the beginning of the list a reference to the last accessible label.

Another problematic situation appeared to be the occurrences of the word **hereby**, which is a synonym to the phrase **thus now**. The problem was that a statement starting with **hereby** can be referred to with **then** or **hence**, but it cannot be labeled because the place of the label should be somewhere between **thus** and **now**. To cope with that, all occurrences of **hereby** were replaced with **thus now**, inserting a label between them.

The same fragment of text from [5] before and after using Delinker is presented in the following example.

Before running Delinker:

```
Lm3:
 for x,A,B,C,D being set holds
  x in A\/B\/C\/D iff x in A or x in B or x in C or x in D
 proof
  let x,A,B,C,D be set;
  hereby assume x in A\/B\/C\/D;
   then x in A\/B\/C or x in D by XBOOLE_0:def 2;
   then x in A\/B or x in C or x in D by XBOOLE_0:def 2;
   hence x in A or x in B or x in C or x in D
      by XBOOLE_0:def 2;
  end;
  assume x in A or x in B or x in C or x in D;
  then x in A\/B or x in C or x in D by XBOOLE_0:def 2;
  then x in A\/B\/C or x in D by XBOOLE_0:def 2;
  hence thesis by XBOOLE_0:def 2;
 end;
```

After running Delinker:

```
R5:
 for x,A,B,C,D being set holds
  x in A\/B\/C\/D iff x in A or x in B or x in C or x in D
 proof
  let x,A,B,C,D be set;
  thus R6: now assume R7: x in A\/B\/C\/D;
   R8: x in A\/B\/C or x in D by R7,XBOOLE_0:def 2;
   R9: x in A\/B or x in C or x in D by R8,XBOOLE_0:def 2;
   thus R10: x in A or x in B or x in C or x in D
      by R9,XBOOLE_0:def 2;
  end;
  assume R11: x in A or x in B or x in C or x in D;
  R12: x in A\/B or x in C or x in D
     by R11,XBOOLE_0:def 2;
  R13: x in A\/B\/C or x in D by R12,XBOOLE_0:def 2;
  thus R14: thesis by R13,XBOOLE_0:def 2;
 end;
```

4. Separator of Library References

Another problem to cope with before the proper experiment was the situation in which the article's author referred to more than one external theorem using the shortened form of reference, e.g.:

```
by XBOOLE_1:2,3;
```

instead of

```
by XBOOLE_1:2,XBOOLE_1:3;
```

This example demonstrates references to the second and third theorems from MML article **XBOOLE_1**. Originally in MIZAR only the full form was possible, so the name of the article had to be repeated. For simplicity, the short form was introduced in which the name of the article appears only once and after the colon numbers of referenced theorems are separated with commas.

Using the short form does not allow for full permutation of references. We can either move all of the combined references together to some other place in the list, or change the order only within themselves. To eliminate this problem, it was enough to write a program which changed all short forms into long ones, which I call Separator of Library References (SEPREF).

Separator of Library References together with Delinker prepared the database of MML articles for the experiment with testing the permutability and monotonicity of references.

5. Software Testing the Permutability of References

Testing the influence of changing the order of references on the results of verifying MIZAR texts required first implementing a program which would alter the order in every reference, and then verifying the new text checking to see if it was still accepted by the system. To carry out the experiment one would have to perform more than $k!$ tests, where k is the maximal number of references in one inference step. A program was written to estimate the number by counting and

reporting the number of references in subsequent inference steps. It turned out that the greatest value of k was 23 – in the article [14]. Therefore, to check all permutations one would need $23!$ tests, which would take approximately $4 * 10^{15}$ years (on a PC with 3 GHz, 2048 MB RAM). This made me think of other ways to study the problem, at least partially, in a much shorter period of time.

The best solution seemed to be writing a program that would generate a random ordering of references in each inference step, and then calling it repeatedly, while at the same time checking the correctness. With this approach, there is a risk that some errors might not be discovered, but with an increasing number of tests the probability of finding new errors converges to zero.

The program PERMREF generates a random ordering of references in each inference step and inserts them in this new order.

6. Software Testing the Monotonicity of References

Testing monotonicity consists of examining whether or not adding redundant references influences the acceptance of statements by the MIZAR *Verifier*. To start this experiment I had to answer three main questions: how many references should be added, what references, and where should they appear.

It seems natural to add only one reference to a straightforward justification or to a statement which is obvious to the system. Adding more references would increase the risk of overflowing some of the limits concerning references.

As to what concerns the kind of references to be added, again the best solution seemed to be based on randomizing, deciding only on the range of values. One method could be to use only references from the current proof block, or the whole current proof, or all accessible statements from the article, or even theorems from MML which are imported by the directive **`theorems`**. The first or second approach would result in a lack of possibilities to choose from in the case of short reasonings. Choosing statements from MML also does not appear to be interesting because it would mostly give statements completely irrelevant to the justified sentence. Therefore, I decided to choose one reference out of all accessible at that point of reasoning within the same article.

Considering the place of inserting the new reference, there are three immediate solutions: at the beginning of the list, at the end, or in a random place. The third option seems to somehow disturb the regularity of inferences, being a special case of permuting. If there is no direct need to use random ordering, I think, it should be avoided. In the case a changing of the order is needed, one may use the program for adding references (MONOTONE) together with the permuting program (PERMREF). Out of the two options, I decided to add randomly generated references at the end of the list.

The program MONOTONE controls additionally whether the new reference is not already on the list. Obviously, it also checks whether or not it is at all possible to find any additional accessible references (in the case that there are no accessible references, the list remains intact).

7. Results of the Experiment

After preparing the MML (using Delinker and Separator of Library References) I ran the permuting program (PERMREF). After only four tests there appeared several articles which were not fully accepted by the MIZAR *Verifier*, but subsequent tests did not result in finding

more errors. The problems concerned the articles [8], [6], [13] and [10]. For example, in [6] the problem looked as follows: **T** denotes a T_0-topological space, where **B** is its basis, **x**, **y** and **X** are arbitrary sets.

```
A1: X in Components(B);
A2: x in X by XBOOLE_0:def 1;
A3: y in X;
```

the following justification was accepted:

```
reconsider x1 = x, y1 = y as Element of the carrier of T
by A1,A2,A3;
```

After permuting – the justification was rejected:

```
reconsider x1 = x, y1 = y as Element of the carrier of T
by A3,A1,A2;
```

The problem was diagnosed and fixed in MIZAR ver. 6.1.14. distributed on November 9, 2002. It concerned the built-in procedure emulating the former theorem **BOOLE:11** which could be written as follows:

```
for x,X,Y be set holds
 x in X & X c= Y implies x in Y;
```

This frequently referenced theorem was built into the *Checker* to enable its use even without a direct reference from the author. The tendency to incorporate such useful facts into the *Checker* in an obvious way strengthens the system and allows for reducing simple background reasoning and at the same time makes them clearer for a potential reader [9]. As the ultimate goal, steps in a system of formalizing mathematics would be similar to those of traditional (pen and paper) mathematics. However, the implementation of **BOOLE:11** in some specific cases resulted in true statements not being accepted by the *Verifier*. What is interesting here is the fact that users of the system had not managed to create such a specific situation, and find the error, for several years. This gives evidence to the fact that studying the robustness of systems for formalizing mathematics is required, or even indispensable, to make them become more useful.

After fixing this problem I ran the experiment another 100 times with MIZAR ver. 6.3.07, and did not find any errors.

Another experiment testing the monotonicity of references consisted in adding random references at the end of each inference, based on MML prepared previously with Delinker. Here also after several repetitions of the experiment there appeared two articles with errors: [4] and [2]. For example, the problem in [4], using the notions below:

```
A,B -- transitive with_units (non empty AltCatStr)
C -- with_units (non empty AltCatStr)
F -- contravariant Functor of A,B
G -- contravariant Functor of B,C
```

looked as follows:

```
A1: G*F is Functor of A,C;
reconsider GF = G*F as Functor of A,C;
A2: GF is covariant;
```

the following justification was accepted:

```
G*F is strict covariant Functor of A,C by A2;
```

After adding **A1** – the justification was rejected:

```
G*F is strict covariant Functor of A,C by A2,A1;
```

The problem was fixed in MIZAR ver. 6.1.16. It concerned the procedure of rounding the mother types in *Equalizer*. *Equalizer* is the part of the MIZAR *Verifier* responsible for equality calculus. Its main role is to join equivalent terms in classes. After the *Equalizer* pass, *Verifier* works not with single terms but classes of equal terms.

The experiment was then repeated 100 times to check whether the problem had been solved. As expected, the procedure reported no errors.

The MIZAR system is continuously developed and it undergoes constant changes and fixes, at the same time new concepts are implemented to strengthen its position among other systems. Therefore it is worthwhile to repeat the experiments from time to time, whenever there is a chance that new errors and problems may appear as a result of changes introduced. So, after some time I decided to re-run the experiments with MIZAR ver. 6.3.07. I repeated testing the permutability 500 times on the whole MML (to minimalise the risk of overlooking some problematic situation). It took 25 days (on a PC with 3 GH and 2048 MB RAM) and no problems were discovered. On the other hand, testing the monotonicity of references was repeated 100 times on the whole MML and showed numerous occurrences of error 4 in the article [3]. One of the contexts is presented below. **C** denotes a category, **S** is a non empty subcategory of **C**, **r1,r2,q2,q3** are objects in **C**, **p1,p2,p3** are objects in **S**, **m1,m2** are arbitrary sets, and finally **q** denotes a morphism of **q2,q3** while **r** is a morphism of **r1,r2**; The theorem **ZFMISC_1:33** states that for any sets **x1,x2,y1,y2** the following holds:

```
theorem :: ZFMISC_1:33
 [x1,x2]=[y1,y2] implies x1=y1 & x2=y2;

A1: [p2,p3]=[q2,q3] & <^q2,q3^><>{} & <^q3,q2^><>{} & m1=q &
q is retraction;
A2: [p1,p2]=[r1,r2] & <^r1,r2^><>{} & <^r2,r1^><>{} & m2=r &
r is retraction;
A3: p2=q2;
```

The following justification was accepted:

```
reconsider m=q as Morphism of r2,q3 by A3,A2,ZFMISC_1:33;
```

After adding **A1** – the justification was rejected:

```
reconsider m=q as Morphism of r2,q3 by A3,A2,ZFMISC_1:33,A1;
```

At the time of this writing, the error has not been fixed yet. The implementation team is still working to fully diagnose and fix the problem.

8. Problems for the Future

Testing the permutability and the monotonicity of references in MIZAR is only the beginning of experiments with the immense issue of testing robustness of systems for formalizing mathematics. The experiments must be repeated from time to time, because (as in the case of testing the monotonicity presented above) quite unexpected results may appear after each change in the implementation, especially when the system evolves very dynamically.

Other issues to be dealt with in the nearest future are finding and eliminating equivalent statements, and then working out methods to delocalize and localize articles (moving statements to the highest or lowest level of reasoning in an article). Combining delocalization with searching for repeated statements may also be interesting, because it should, at least theoretically, give stronger results than without delocalization. Further tasks will be assigned after finalizing the research presented here. Although in general the issue of robustness is immense and cannot be fully solved, any system which undergoes such experiments has a chance to become truly robust system.

9. Acknowledgments

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

References

1. Ewa Bonarska: An Introduction to PC MIZAR, Fondation Ph. le Hodey, Brussels, 1990
2. Zbigniew Karno: Maximal Kolmogorov Subspaces of a Topological Space as Stone Retracts of the Ambient Space, Formalized Mathematics, 5(1), 1996, pp. 125–130
3. Artur Korniłowicz: On the Categories Without Uniqueness of cod and dom. Some Properties of the Morphisms and the Functors, Formalized Mathematics, 6(4), 1997, pp. 475–481
4. Artur Korniłowicz: The Composition of Functors and Transformations in Alternative Categories, Formalized Mathematics, 7(1), 1998, pp. 1–7
5. Artur Korniłowicz, Robert Milewski: Gauges and Cages. Part II, Formalized Mathematics, 9(3), 2001, pp. 555–558
6. Robert Milewski: Components and Basis of Topological Spaces, Formalized Mathematics, 9(1), 2001, pp. 25–29
7. Michał Muzalewski: An Outline of PC MIZAR, Fondation Philippe le Hodey, Brussels, 1993
8. Adam Naumowicz: On Segre's Product of Partial Line Spaces, Formalized Mathematics, 9(2), 2001, pp. 383–390
9. Adam Naumowicz, Czesław Byliński: Basic Elements of Computer Algebra in MIZAR, Mechanized Mathematics and Its Applications, 2, 2002, pp. 9–16
10. Adam Naumowicz, Mariusz Łapiński: On T_1 Reflex of Topological Space, Formalized Mathematics, 7(1), 1998, pp. 31–34
11. Piotr Rudnicki: An Overview of the MIZAR Project, Proceedings of the 1992 Workshop on Types for Proofs and Programs, Chalmers University of Technology, Bastad, 1992
12. Piotr Rudnicki, Andrzej Trybulec: On Equivalents of Well-foundedness. An experiment in MIZAR Journal of Automated Reasoning, 23, pp. 197–234, Kluwer Academic Publishers, 1999
13. Andrzej Trybulec: A Borsuk Theorem on Homotopy Types, Formalized Mathematics, 2(4), 1991, pp. 535–545
14. Akihiko Uchibori, Noboru Endou: Basic Properties of Genetic Algorithm, Formalized Mathematics, 8(1), 1999, pp. 151–160
15. Freek Wiedijk: Comparing Mathematical Provers, Proceedings of the Second International Conference on Mathematical Knowledge Management, LNCS 2594, Springer-Verlag, 2003, pp. 188–202

A Note on Translating Mizar Articles into *OpenMath*–Related Formats

Markus Moschner

Dept. of Computer Science (Dept. of Statistics), University of Vienna, Austria.

Abstract – Due to its large library of mathematical content, *Mizar* has received attention as an interesting system for mathematical knowledge representation. This raises interest of utilization of its content by other systems (and vice versa). An effort on translation of *Mizar* articles to *OMDoc* format is reported. Some reflections on the experiences and problems herewith are given.

1. Introduction

There have been already a lot of efforts to deal with (or manage) mathematical texts before the young discipline of *mathematical knowledge management (MKM, [JD01])* was founded. As is typical for the field of computer research and practice, much was going on without respect for compatibility (maybe not always without certain intentions). It is no surprise that systems like *Mizar* has little or no connection to other systems. Hence translation issues come up in the course of times for various reasons such as: usage of different strengths, comparability or simply content exchange.

Mizar[WWW] has received strong attention within the *MKM* community. Texts written in *Mizar* are close enough to common mathematical vernacular to be readable without much difficulties by mathematically educated people. Such properties cannot be seen as granted till today — only for pure presentation systems which are not in focus here, though there are other powerful systems around for checking mathematical or logical content. Consequently, the question arises of how to get content from *Mizar* stuff somewhere else or vice versa. Basic knowledge of *Mizar* will be assumed.

1.1 *OpenMath*

The desire of computational information exchange led to demands of unambiguous representation of mathematical content. *OpenMath* ([OMS]) is an effort to fulfill such needs for the field of mathematics. It is a language for the representation and communication of mathematics. Planned as a special purpose language for computer algebra systems, its development resulted in one for mathematics in general and logics. Consequently, *OpenMath* can be used for expressing formal mathematical objects appropriate for todays theorem provers and proof checkers.

Mathematical symbols get defined in so called *content dictionaries*(CD). The *OpenMath* society runs a list of CDs for various purposes. Moreover, it defines standards for *OpenMath* (including the structure of OpenMath objects, encodings and content dictionaries).

MathML has a similar conceptual background with strong observance for WWW–usability (coverage of the presentation issue); aside from some close connections with *OpenMath* (and *OMDoc*) it is a W3C recommendation which is implemented in todays prominent browsers (e.g., *Mozilla* and *Amaya*).

Manuscript received August 17, 2004; revised October 15, 2004

1.2 *OMDoc*

OMDoc (**O***pen Markup Format for* **M***athematical* **Doc***uments*,[Ko00]) aims at flexible markup for mathematical content beyond *OpenMath*. While *OpenMath* (and *MathML*) only care for special formalizing of mathematical content, semantic flexibility is a fundamental issue where *OMDoc* goes beyond *OpenMath* and *MathML*. Some features of *OMDoc*:

- Semantical foundations are definable, thus giving the possibility of different underlying logic systems.
- Presentation and content are separated (multiple presentations allowed).
- Formal (FMP) and informal (CMP) contents are distinguished. The first acts as the machine–readable content.
- The core mathematical items are as usual — symbols, definitions, assertions, axioms and proofs.
- Mathematical items carry metadata information (*DublinCore*).

FMP contains *OpenMath* objects. Yet FMP may be empty.
MBase[FK00] is a structured repository of *OMDoc* documents.

2. Method of Approach

The language *Mizar* is not really small for today's customs (in [CG04],p.63: 93 keywords and 150 productions). Restriction to abstracts saves about 20 keywords defined only for proofs. Since the crucial problems are not bypassed through such a decision that seems a justifiable restriction for the first prototypes (and abstracts are covered already): proofs consist essentially of formulas with justifications, the treatment of formulas has to be solved already within abstracts, justifications are clearly unambiguous.
Thus the aim of translating a *Mizar* article is split into these successing sub–goals:

1. metadata generation
2. transformation of mathematical content structure
3. formula translation
4. (proof translation —
   however that is some sort of replication of steps 2–3)

Furthermore a *Mizar* article is split into its tree structure down to atomic terms. Programming is done in ML ([Er99]). The implementation used is *OCaml* ([CMP00]), yet the code uses only *Caml* statements. Though the choice had less methodological reasons, one should not ignore the fact that this implementation is a well developed full purpose language with a rather large amount of (parsing) tools. *MBase* is also used as a testsuite for *OMDoc* files.

3. On the Translation

The content of MML contributions is scattered across more than one file (bibliography, vocabulary and article). In some senses, that resembles the "old scientist's" approach of keeping different classes of content in different locations (be it just only different chapters in a book). Nevertheless, programming engineers seem strongly opposed that view. They want files in WYSIWYG–style with everything included in one file — the attitude behind XML (and *OMDoc*).

3.1 Starting with Metadata

Vocabularies are needed only for processing articles and the bibliography does not interfere with the content of an article hence that information may be merged into one file rather straightforwardly. The few main issues:

1. Single *Mizar* articles should correspond with single *OMDoc* theories. Though that need not be the case (each single *Mizar* text–item could be given its own theory or the whole MML could be crunched into one theory). It seems to be the best complaisance for the authors of articles.
2. Bibliographical data are obviously metadata in *OMDoc* with the exception of abstracts and references. There is no appropriate *DublinCore* field for these things.
3. A *Mizar* vocabulary is not really needed with respect to content. Yet priority information shall not be lost what justifies one direction (see table below). Since names of vocabularies do not correspond in a standardized way to the names of articles, there is no proper correspondence for a backwards direction.

With respect to the general picture above, the points can be summarized as follows:

Mizar entity		*OMDoc* item
article	$\Longleftrightarrow$	*theory*
bibliography	$\Longleftrightarrow$	*metadata of theory*
vocabulary	$\Longrightarrow$	*symbol* definitions (for *Mizar* definition items)

Priorities of functors is part of the information within vocabularies which is needed for symbols. Such information is not available only from the article in question. Its relevance is not only for further translation issues, but also for presentation (of the *OMDoc* translation, which is no *Mizar* specific matter any more).

3.2 Diving into the Proper Text

Mizar is minimalistic in the sense that there are three main classes of mathematical “items” (similar to informal mathematics): definitions, theorems and proofs. *OMDoc* reflects roughly this distinction yet with refinement by attributes for elements. The general picture may be summarized analogously to the table of the foregoing section:

Mizar item		*OMDoc* item
environment	$\Longrightarrow$	import/catalog elements
section	$\overset{?}{\Longleftrightarrow}$	*omgroup*
definition	$\Longleftrightarrow$	*definition*
excl. *cluster/registration*	$\Longrightarrow$	*assertion*
structure	$\Longleftrightarrow$	*definition*
scheme	$\Longleftrightarrow$	*definition*
redefinition	$\Longleftrightarrow$	*alternative*
theorem	$\Longleftrightarrow$	*assertion*
proof	$\Longleftrightarrow$	*proof*

Some remarks on that table:

1. *OMDoc* includes mechanisms for theory inheritance, yet not so fine–grained as *Mizar*.

2. *Mizar sections* correspond intuitively with *OMDoc omgroups* because of their usability for grouping inside *theory* elements. On the other hand, the sectioning in *Mizar* seems to be used only for (journal) presentation what is not of much use for machine–oriented translation.
3. *Modes* as one of the definition items introduce new *Mizar types*. In a full translation, *Mizar types* are a part of the formal content in definitions (also of new notions). These cannot have fully–fledged definitions. One can follow [Ur03b] (p.204) and introduce extra definitions — that point does not refer to first–order translations only. Preference should be given to the first approach mentioned there. Since set–theoretic classes commensurate better the common constructions of sorts (and types), especially if these are used for bounding of quantifiers.
4. Clusters (now registrations) in *Mizar* are obviously theorems. Authors choose such clusters if they think them to be of importance or of need. Yet not each existential *assertion* in *OMDoc* should get a role as a cluster. At least, such practice will look very strange contrariwise to the "common *Mizar* style".
5. *OMDoc* allows definitions with recursive or implicit formulations (including proofs as parts of definitions similar to *Mizar schemes*). Hence, there is no expressibility problem per se.
   One should note that there are multiple proofs for each theorem allowed.
6. A less innocent topic concerns reconsiderations (in proofs). Since reconsiderations have to be justified these could get sub-proofs or local lemmata (before the theorem in question). These are quite often *Mizar type* changes which are tangent to the treatment of formal content.

As yet structure is only concerned in such a way that context-free parsing would hardly find any stumbling blocks. *OMDoc* allows for putting item blocks into *CMP* (*commented mathematical property*) which is thought for informal text. Such content is not available (and not intended) for mechanized communication. *OpenMath* objects may still be used inside, yet their relevance rests only on presentation and as text. This is useful for similar text generation as the "*Mizar* journal" [JFM]. In principle, the translation of *Mizar* articles to its JFM version could be stored inside of *CMPs* in a "parallel" way.

3.3 Tackling Formal Content

The main interest into translations concerns formulas since these represent the formal content of an article (so far TEX–content could be processed similarly). It is the place where the problems of context sensitivity come into play. Formal content in *OMDoc* is represented as *OpenMath* content: notions are indicated by references to its definitions (in other theories). *OpenMath* CDs (content dictionaries) already exist for classical logics and basic mathematical notions. Thus, the main problem is getting the references for the atomic parts of formulas — the peculiar business of the *Mizar* parser. Because the same designators may be (and are) defined differently in different *Mizar* articles, this can get rather tricky (as pointed out in [CG04], sect 4). That and especially overloading are doubtlessly one of the charming properties of *Mizar*. In the sense of [Ur03b] (sect.2.2) a *Mizar* article gives sight into the "pattern level", but what is needed is the "constructor level".

If such things are not available things turn unfavorable. The WWW–abstracts already contain references and can be helpful in this way — especially for *Mizar* independent tools (the problem of proofs keeps open). Another source is a group of special files originally generated by Bancerek for use by MML–*Query*. These contain (vital) information which is not available by the MML–*Query* tool at now. With such files there is a possibility to translate single files (of

MML) — without a labour expenditure similar to the reimplementation of the *Mizar* parser. The only alternative would be some sort of bootstrapping — starting with the most basic articles and ascending a hierarchy which is available from the environments.

Some further technical issues:

- The "naked" mathematical (formal) content itself is of interest — either by technical reasons or by need of natural look. *Mizar* seems to be appealing due to somehow reflecting the use of sorts in mathematical practice (yet without much care about the formal notion of sorts). In these circumstances, there is not much lack of types, either.
  Accordingly, a bunch of *OMDoc* files (used by Ω*mega* as example) make no use of its ability in type representation.
  Hence, translations need not take into account typed formal content in *OMDoc*. Still use of the *Mizar* type information remains a topic, since *Mizar* types resemble sorts which means a bounding of quantifiers. In addition to the remark on modes in the above subsection, a set–theoretic approach for translation complies with such an analysis — though that is contrariwise to Urban's decision ([Ur03b]).
- In the above sense, *mode* definitions may be stated as introductions of non–empty classes with certain properties.
- In the above sense, reconsiderations may be formulated via elementhood.
  A type–oriented approach would need some extra formulation of axioms on (*Mizar*) types inheritance. [Ba03] deals with properties on *Mizar* types. Such properties reflect the way *Mizar* is working — this means additional axioms through its formalization.
- *OMDoc* prohibits ambiguity fundamentally. Overloading cannot be expressed directly. There have to be different unique notions (maybe with the same *commonname* — the trivial name in *OMDoc*) which have the same presentation — that amounts to the same look for a human viewer but the machine–readable formal content is distinct. A justification for overloading is human readability. Whether this reflects the intention of overloading sufficiently needs some further considerations.

4. Discussion and Conclusion

Mathematical content cannot be analyzed without its context. Close connections between semantical meaning and structuring characterize mathematically non–simple text. Such peculiarities cannot be spirited away by translation. That requires for *Mizar* articles the same work as the *Mizar* system in matters of parsing (yet parsing here cannot be the context–free ticket of usual programming languages). Certain ("readability") styles of writing *Mizar* articles can make things easier — such restrictions would not really seem a deprivation since articles should be "readable". As mentioned in [CG04] third parties do not find open doors for translating *Mizar* content. What has to be done first is more or less not only the business of *Mizar* itself, moreover *Mizar* needs this before checking. This seems to be one of the central parts of *Mizar* to me. The effort would result in some sort of reimplementation, and reinventing the wheel is not always recommendable. Maybe a better solution is an interface with *Mizar* parser output which could be the starting point for translations. It seems that [Ur03a] and [BR03] already build upon some sort of forerunner(s) for such a task.
Especially [BK03] is based on such MML–*Query*–related tools. The authors have done some partial translations from *Mizar* to *OMDoc*. Steps 2. and 3. of Section 2 are the main focus there. Complete sentences or formula–expressions of text items are queried against an MML–*Query*–related tool which produces a translation to *OpenMath*–format.

As already mentioned the *Mizar* language is in a certain sense minimalistic: no lemmata, propositions, subproofs (as notion, of course as iterations). Thus one needs only a proper subset of markup languages like *OMDoc*. On the other hand there are some *Mizar* things which have no (good) counterpart in *OMDoc* (*environment*, *registration*, e.g.) — and that counts for similar markup languages, too. Hence, one can use only a proper subset of markup languages like *OMDoc* — yet with more flexibility. It seems that smaller (or: lighter) markup languages could ease the burden of cross–translating (for each side). That could give way for plainer insights into the needs of mathematical markup.

4.1 Retranslations?

For practical reasons newer *OMDoc* versions allow for the inclusion of something analogous to *verbatim* in TEX, so that *OMDoc* databases (like *MBase*) do not parse content that is specific for special theorem provers or checkers — and needed for retranslation purposes. Elements for informal content would be rather impractical to handle. For a "backtranslation" of unchanged articles some artificial things in *OMDoc* need to be done at the translation step — with the hope that confusion has a very low possibility. To get the articles for an environment is not really the problem (at least for *MBase* because of the inheritance mechanism). However, the structuring of an environment can get tough. Due to combinatorial explosion, brute force seems rather inadequate. There are no special metadata constructs in *OMDoc* (or *MathML*) available for such purposes. It seems that mathematicians underestimate the role of metadata for guidance — these are more than author information for example. Another good source of problems are the already mentioned clusters and modes. In particular, modes are defined arbitrarily by an author. New modes introduced have then to be recognized in the following text. Hence, methodology for (semi–)automated introduction of modes has to be based upon a mechanism with well developed semantic capabilities — something that seems not to have satisfying solutions today.

Retranslation need not be seen only with respect to full articles — some remarks with respect to parts of an article:

1. Predicate logic reasoning:
   Mizar authors still need to work out proofs for tautologies which are not very complicated; there are really good theorem provers around just for such issues.
2. Equation handling:
   The *Mizar* checker is sometimes to tough and gives an unprovability error, if not necessary; what surely ensures correctness is not always an author's joy; there are rather good provers around, too.

4.2 Conclusions and Further Work

Work on translations can give insight into differences between existing systems. Furthermore this can be used for a comparison of different approaches on special topics (as an example there are different approaches to category theory in *Mizar*). For practical insights, the approach of Urban [Ur03b] bears on theorem prover formats. Such formats are the result of evolvement, why these have developed with needs and implementations. Usability for *Mizar* will be more likely this way, especially if one expects some saving of tedious steps in *Mizar*, when an author mainly simulates a predicate logic reasoner. Another aspect is that the *Mizar* developers do not want to change a working system (at least not too much, which is an honourable attitude). Efforts on retranslation open the possibility for inclusion of "foreign" trustable results without endangering a well grown up system.

OMDoc arose mainly of academic interest. One main aim behind it was the production of a rich format for many notions of mathematics. That classifies it as SGML within mathematical markup languages (do not take this analogue too literally — as expressibility does not go beyond XML).
Such an isolation of construction seems to indicate a lack of interfaces for existing theorem provers or checkers. At least there exist some for theorem provers of higher–order logic which are of special interest for *Mizar*. One may not forget that *OMDoc* cares for representation of its content, and *MBase* can be used as a search and exchange tool. Though the *Mizar* community will have more interest in retranslations, it is a far way to go still. Restriction to predicate logic reasoning (as mentioned above) could be an acceptable compromise at first.

Completion of *Mizar*–to–*OMDoc*–translation for abstracts has still to be developed to a satisfiable state. There are some issues with more than one possibility of translation. So different styles of translation might be followed. Retranslation has still to be surveyed. For being a very (tedious) technical matter one can hope that translation issues give insights and clues on *MKM* topics as design of systems as well as content representation, and that they can help in the understanding of the crucial differences between systems or representation methods.

Acknowledgments

The author acknowledges the Calculemus IHP Network FP5 EU Programme (HPRN-CT-2000-00102) for partial support of this work.

References

[ABD03] A. Asperti, B. Buchberger, J.H. Davenport (eds.): Proc. of the 2nd Int. Conf. on Mathematical Knowledge Management; Springer–Verlag, 2003, LNCS 2594.

[Ba03] G. Bancerek, On the Structure of Mizar Types, in: H. Geuvers and F. Kamareddine (eds.), Proc. of MLC 2003, ENTCS 85(7), 2003.

[BK03] G. Bancerek, M. Kohlhase: Towards a Mizar Mathematical Library in OMDoc Format; Electronic Proceedings of the Workshop Mathematics on the Semantic Web, Eindhoven, The Netherlands, 2003.

[BR03] G. Bancerek, P. Rudnicki: Information Retrieval in MML; in [ABD03], p.119–132.

[CG04] P. Cairns, J. Gow: Using and Parsing the Mizar Language; Electronic Notes in Theoretical Computer Science, 93, 2004, pp.60–69.

[CMP00] E. Chailloux, P. Manoury, B. Pagano: Developing Applications With Objective Caml; O'Reilly, Paris, 2000 (prelim. translation of *Développement d'applications avec Objective Caml*).

[JD01] J. Davenport; Mathematical Knowledge Representation; in: B. Buchberger, O. Caprotti (eds.), Proceedings of MKM 2001, Linz, Austria, 2001.

[Er99] M. Erwig: Grundlagen funktionaler Programmierung; Verl. Oldenbourg, Munich, Germany, 1999.

[FK00] A. Franke, M. Kohlhase: *MBase*: Representing Knowledge and Context for the Integration of Mathematical Software Systems; Journal of Symbolic Computation; Special Issue on the Integration of Computer Algebra and Deduction Systems, 32(4),2001, pp. 365–402.

[JFM] Journal of Formalized Mathematics, **http://mizar.org/JFM**

[Ko00] M. Kohlhase: *OMDoc*: An Infrastructure for OPENMATH Content Dictionary Information; Bulletin of the ACM Special Interest Group on Symbolic and Automated Mathematics (SIGSAM), 34(2),2000.

[W3C] W3C Math Home, **http://www.w3.org/Math**

[WWW] The Mizar Home Page, **http://mizar.org**.

[MML] The Mizar Mathematical Library, **http://mizar.org/library**

[OMS] The Openmath Society, **http://www.openmath.org**

[Ur03a] J. Urban: MPTP 0.1; in: I. Dahn, L. Vigneron (eds.), Proc. of FTP 2003, ENTCS, 86(1), 2003.

[Ur03b] J. Urban: Translating *Mizar* for First Order Theorem Provers; in [ABD03],p.203–215.

MECHANIZED MATHEMATICS AND ITS APPLICATIONS, VOL. 4, NO. 1, MARCH 2005, 67:74

Formalization of Commodity Space and Preference Relation in Mizar

Krzysztof Wojszko and Artur Kuzyka

Department of Computer Science, Białystok Technical University, Białystok, Poland

Abstract – The goal of this work [WK] is to formalize Elements of Consumer Demand Theory in the Mizar language. The theory is based on the book: "Elements of mathematical economy" by E. Panek. This is not a direct formalization, but a generalization of the theory. In the formalization we try to generalize the theory and fill recognized gaps.

1. Introduction

In this paper we describe how commodity space and preference relation can be formalized in the Mizar system. This is the first article in Mizar related to this economic subject. Some economic aspects were also formalized in Mizar by Krzysztof de Werszowec Rey [WR93]. Our formalization introduces basic definitions and theorems, which can be used further for proving more advanced theorems. We hope our article initiates a new branch in Mizar and will make possible check economical models. We focus on the following elements of commodity space: basket of commodities, basket neighborhood, indifference and preference relation.

2. Commodity Space

From among all available commodities on the market a consumer can pick a bundle of them. This bundle is called a baskets of commodities and all of them are included in the commodity space. According to Panek, a commodity space is a subset X of $\Re^n_+$ which contains all available baskets of commodities equipped with distances between baskets. For baskets there are also defined operations of addition and multiplication by non negative real numbers similar to operations in vector spaces. Theories of metric spaces and vector spaces have been already developed in Mizar and we decided to use them for our purpose. First we define the structure of a vector space with a distance function (in Mizar: **MetrVectStr**) which merges **VectSpStr** [KLM90] and **MetrStruct** [KLS90]. Then the structure of the commodity space is defined as **MetrVectStr** with new field **baskets** which is a subset of the **carrier** as shown below. The concept of Mizar structures is well described in [RT99].

```
definition
  let F be 1-sorted;
  struct (MetrVectStr over F) CommodityStr over F (#
    carrier -> set,
    baskets -> Subset of the carrier,
    distance -> Function of [:the carrier,the carrier:],REAL,
    add -> BinOp of the carrier,
    Zero -> Element of the carrier,
    lmult -> Function of [:the carrier of F,the carrier:],
```

Manuscript received August 22, 2004; revised October 18, 2004

```
                              the carrier
    #);
  end;
```

To define commodity space we introduce the attributes **void**, **positive**, and **with_empty_basket**, which mean respectively that the set of baskets is empty, baskets are in "positive area", and empty basket (the **Zero** element) is in the set of baskets. In our general approach, it is impossible to define positive area. We avoid it by defining **positive** by the condition that **Zero** is the only basket whose opposite vector is also a basket.

```
definition
  let C be non empty CommodityStr over F_Real;
  attr C is positive means
    for x being Element of C
      st x in the baskets of C & -x in the baskets of C
    holds x = the Zero of C;
end;
```

The attributes determining conditions for vector spaces (**VectSp-like**) and metric spaces (**Reflexive**, **discerning**, **symmetric**, **triangle** [KLS90]) are also used in our formalization. These conditions seem to be not enough to define commodity space, as the structure of a set of baskets cannot be fully described. So, we introduce a *quasi* commodity space:

```
definition ::CommoditySpace
  mode PreCommoditySp is
    positive with_empty_basket VectSp-like complete
    (non void Reflexive discerning symmetric triangle
     CommodityStr over F_Real);
end;
```

which has to satisfy axioms of metric space (in our case they became Mizar theorems) shown bellow:

```
theorem Th3:
  dist(x,y)>=0 by METRIC_1:5;

theorem Th4:
  dist(x,y)=0 iff x=y by METRIC_1:2,1;

theorem Th5:
  dist(x,y)=dist(y,x);

theorem Th6:
  dist(x,y)<=dist(x,z)+dist(z,y) by  METRIC_1:4;
```

convergence of Cauchy sequences (**complete**), and axioms of vector spaces.

As a test of correctness of this approach, we prove the following theorems from the book [Pan02] related to topological properties of a *quasi* Commodity Space. We introduce some topological notation missing in MML:

- **closed, open, compact** as attributes for subsets of a metric space (in our case, for sets of baskets),
- **R-neighborhood(x)** as a synonym for an open ball with center in **x** and radius **R**,
- predicate **is_internalpoint_of** and operation **Int**,
- predicate **is_boundarypoint_of** and operation **Bnd**.

```
definition
  let X be non empty MetrSpace;
  let A be Subset of X;
  attr A is closed means A is closed Subset of TopSpaceMetr X;
  attr A is open means A is open Subset of TopSpaceMetr X;
  attr A is compact means A is compact Subset of TopSpaceMetr X;
end;
```

To test the correctness of the attribute **closed** we prove the characterization of it in terms of convergence (of Cauchy sequences):

```
theorem Th8:
  M is closed iff for SB being sequence of C
  st SB is convergent holds lim SB in M
```

The attribute **open** is tested by the characterization of internal points. Namely, a set M is **open** if every basket $x \in M$ is one of its internal point:

```
theorem
  M is open iff for x st x in M holds x is_internalpoint_of M;
```

Eventually, a set M is **compact** if it is **bounded** and **closed** where a set M is **bounded** if there exists a positive number N such that the distance between any two baskets in this set does not exceed N (see below)

```
theorem
  M is bounded iff ex R st R > 0 & for x,y
  st x in M & y in M holds dist(x,y) <= R by TBSP_1:def 9;

theorem Th12:
  M is compact iff M is bounded closed;
```

A basket is an internal point of a set M if it has an open neighborhood included in M.

```
definition
  let C be MetrStruct, x be Element of C, M be Subset of C;
  pred x is_internalpoint_of M means
    ex R st R > 0 & R-neighborhood(x) c= M;
end;
```

The interior of a set M is the set of all of its internal points.

```
definition
  let X be non empty MetrSpace;
  let A be Subset of X;
```

```
  func Int A -> Subset of X equals
    {x where x is Element of X: x is_internalpoint_of A } ;
  coherence
end;
```

A boundary point of a set M is a basket from M with each open neighborhood having a common point with the complement of M (**M`**).

```
definition
  let C be MetrStruct, x be Element of C, M be Subset of C;
  pred x is_boundarypoint_of M means x in M & for R st R > 0
    holds   R-neighborhood(x) meets  M`;
end;

definition
  let X be non empty MetrSpace;
  let A be Subset of X;
  func Bnd A -> Subset of X equals
    {x where x is Element of X: x is_boundarypoint_of A } ;
  coherence

end;
```

Please note that **Bnd A** differs from **Fr A** from [WD90]. For example a boundary of open closed segment $S = (0,1>$ is different than the frontier of this segment. $Bnd(S) \neq Fr(S)$

Next, to test vector space aspects of *quasi* commodity space we investigate the theory of convexity. Our goal is to define commodity space as a *quasi* commodity space with a convex set of all baskets. In such spaces, a linear convex combination of baskets is also a basket. Linear convex combination of baskets x, y is a combination $\alpha x + \beta y$, where α and β are non negative real numbers such that $\alpha + \beta = 1$.

```
definition
  let C be PreCommoditySp;
  let x,y;
  mode LinearConvexCombination of x,y -> Basket of C means
  ex R1, R2 being non negative Real st
    it = R1 * x + R2 * y & R1 + R2 = 1;
  existence;
end;
```

Set M is convex if all linear convex combinations of any two baskets included in M also are included in M:

```
definition
  let C be PreCommoditySp;
  let M;
  attr M is convex means :defCon:
  for x,y st x in M & y in M for R1, R2 being non negative Real
  st R1 + R2 = 1 holds R1 * x + R2 * y in M;
end;
```

3. Preference Relation

A consumer exposes his preference by comparing two commodities or baskets of commodity. The consumer may decide that one is better than the other (preference relation) or they are equally preferred (indifference relation). To formalize that, we define **PreferenceStr** which contains the **carrier** and two relations on this **carrier**. In MML the theory of ordered sets was already well advanced [BR02] (based on the structure **RelStr**). Because the preference relation is similar to the ordering relation we decided to define **PreferenceStr** as an extension of **RelStr**.

```
definition
  struct (RelStr) PreferenceStr (#
    carrier -> set,
    InternalRel, IndiffRel -> Relation of the carrier
  #);
end;
```

The relation **InternalRel** corresponds to a preference relation and **IndiffRel** corresponds to an indifference relation of consumer.

For convenience and to assimilate notation to that from Panek's book [Pan02], we introduce some definitions . Therefore we can write

- **x Succ y**,[1] which means **x** is preferred over **y**,
- **x prec y** means **y** is preferred over **x**,
- **x sim y** means **x** is indifferent from **y**,
- **x succsimm y** which means **x** is preferred over **y** or **x** is indifferent from **y**,
- **x precsim y** analogically.

```
notation
  let P being PreferenceStr;
  let x,y be Element of P ;
  synonym x succsim y for x <= y;
  synonym y precsim x for x <= y;
end;

definition
  let P being PreferenceStr;
  let x,y be Element of P ;
  pred x sim y means :defSim: [x,y] in the IndiffRel of P ;
end;

definition
  let P being PreferenceStr;
  let x,y be Element of P ;
  pred x Succ y means :defSucc: x succsim y & not x sim y;
end;
```

[1] \succ in TeX is rendered as $\succ$, \sim as $\sim$, \prec as $\prec$, \succsim as $\succsim$, and \precsim as $\precsim$.

According to the book [Pan02] a preference relation has to satisfy the following axioms:

- connectedness: $\forall x,y \;\; (x \precsim y) \vee (y \precsim x)$
- transitivity: $\forall x,y,z \;\; (x \precsim y) \wedge (y \precsim z) \Rightarrow x \precsim z$

So we define a preference relation as a **connected** and **transitive** relation.

```
definition
  let C be non empty 1-sorted;
  mode Preference of C is
    connected transitive Relation of the carrier of C;
end;
```

Formalization of further properties of the preference relation has shown that the strong part of it ($\prec = \precsim \setminus \sim$) should satisfy the axiom

- asymmetry: $\forall x,y \;\; x \succ y \Rightarrow \neg(y \succ x)$

Without such an assumption some theorems from the book [Pan02] would be false (see point b) below).

The indifference relation is defined in the book [Pan02] as a set of all pairs of baskets which are indifferent, and the following conditions are satisfied:

- reflexivity: $\forall x \;\; x \sim x$
- symmetry: $\forall x,y \;\; (x \sim y) \Rightarrow (y \sim x)$.

To formalize it we used **Tolerance** relation, which is **total**, **reflexive**, and **symmetric** already defined in MML. So the **Indifference** is just a **Tolerance** relation:

```
definition
  let C be 1-sorted;
  mode Indifference of C is
    Tolerance of the carrier of C;
end;
```

Finally Consumer Preference is defined as **PreferenceStr** such that an indifference relation is included in the preference relation and a strong preference relation is asymmetric.

```
definition
  mode ConsumerPreference -> non empty PreferenceStr means :defPP:
    the InternalRel of it is Preference of it &
    the IndiffRel of it is Indifference of it &
    the IndiffRel of it c= the InternalRel of it &
    (the InternalRel of it) \ the IndiffRel of it is asymmetric;
end;
```

Under this condition we have the following theorems:

a) $x \succ y \vee y \succ x \vee x \sim y$ which is formalized as:

```
theorem
  for P being non empty ConsumerPreference;
  for x,y being Element of P holds
    x Succ y or y Succ x or x sim y;
```

b) $(x \succsim y \wedge y \succsim x) \Leftrightarrow x \sim y$

```
theorem
  for P being non empty ConsumerPreference;
  for x,y being Element of P holds
    ( x succsim y & y succsim x ) iff x sim y;
```

In the proof of this theorem the asymmetry of strong preference was necessary.

c) $(x \succsim y \wedge \neg(y \succsim x)) \Leftrightarrow x \succ y$

```
theorem
  for P being non empty ConsumerPreference;
  for x,y being Element of P holds
    ( x succsim y & not ( y succsim x ) ) iff x Succ y;
```

d) $(x \succsim y \wedge y \succ z) \Rightarrow x \succ z$

```
theorem
  for P being non empty ConsumerPreference;
  for x,y,z  being Element of P holds
    ( x succsim y & y Succ z ) implies x Succ z;
```

4. Consumer Preference Filed

As a final result we defined a consumer preferences field. It is **ConsumerPreference** concerning the set of baskets of a commodity space. To define this, we had first to merge structures of commodity space and consumer preferences (**PreferenceFiledStr**). Finally, consumer preferences fields are formalized as:

```
definition
  mode PreferenceField -> PreferenceFieldStr over F_Real means
    it is CommoditySp & it is ConsumerPreference;
end;
```

References

[BR02] Grzegorz Bancerek and Piotr Rudnicki. Compendium of continuous lattices in mizar. *Journal of Automated Reasoning*, 29(3-4):189–224, 2002.

[KLM90] Eugeniusz Kusak, Wojciech Leończuk, and Michał Muzalewski. Abelian groups, fields and vector spaces. *Formalized Mathematics*, 1(**2**):335–342, 1990.

[KLS90] Stanisława Kanas, Adam Lecko, and Mariusz Startek. Metric spaces. *Formalized Mathematics*, 1(**3**):607–610, 1990.

[Pan02] E. Panek. *Elements of mathematical (in Polish).* Poznan : Wydawnictwo Akademii Ekonomicznej w Poznaniu, 2002.

[RT99] Piotr Rudnicki and Andrzej Trybylec. On equivalents of well-foundedness. *Journal of Automated Reasoning*, 23(3-4):197–234, 1999.

[WD90] Mirosław Wysocki and Agata Darmochwał. Subsets of topological spaces. *Formalized Mathematics*, 1(**1**):231–237, 1990.

[WK] Krzysztof Wojszko and Artur Kuzyka. Formalization of commodity space and preference relation in mizar. http://www.wkw.w.tkb.pl/mizar.

[WR93] Krzysztof Werszowec-Rey. *Formalizacja Badan Spolecznych.(wspomagana komputerowym systemem Mizar kontroli poprawnosci logicznej) (in Polish)*. Fundacja im. Philippe le Hodey, 1993.

MECHANIZED MATHEMATICS AND ITS APPLICATIONS, VOL. 4, NO. 1, MARCH 2005, 75:82

On the Mizar Encoding of the Fibonacci Numbers

Adam Grabowski and Magdalena Jastrzębska*

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

Abstract – In this paper we describe the development of the Fibonacci numbers and their generalizations (e.g., Lucas and the so-called G–numbers) in the Mizar Mathematical Library. We focus on the proof of the Carmichael theorem on the prime divisors of prime-generated Fibonacci numbers – based on the observation of Shane Findley made only a few years ago.

1. Introduction

The idea of Fibonacci numbers is dated back to 1202, when Leonardo Pisano (Fibonacci) introduced in his book *Liber abaci* the problem "how fast could rabbits breed in ideal circumstances?" The name of the series obtained in this way appeared in the present form much later about 1870 and is due to French mathematician Edouard Lucas.

Although the sequence we deal with was introduced rather early in Mizar (about 10 years ago), it was not really developed in the Mizar Mathematical Library, at least to the extent it deserves. Our mission was to formalize main facts concerned with the Fibonacci numbers to create a sort of monograph. We decided also to choose a fact which is relatively well-known with rather recent proof. The theorem we considered was the Carmichael Theorem.

This is a description of the work already done in Mizar (see [4], [8] for full Mizar representation).

In Section 2 we briefly describe how the Fibonacci sequences are defined in Mizar and point out some improvements based on the schemes for recursive definitions in Section 3. The next three sections contain the Mizar encodings of the set of basic identities, the restricted Carmichael Theorem, and the connections of Fibonacci numbers with Pythagorean triples, respectively. Section 7 points out three feasible generalizations/modifications of Fibonacci sequences, and in the last section we draw some concluding remarks and discuss our plans for future work.

2. Fibonacci Numbers and Their Formalization

Fibonacci-type sequences are probably most famous for their recursive definition. The classical one is defined just as

$$Fib(n) = \begin{cases} 0, \text{ if } n = 0, \\ 1, \text{ if } n = 1, \\ Fib(n-1) + Fib(n-2), \text{ otherwise}, \end{cases} \tag{1}$$

* The work described here is a subset of the second author's B.Sc. thesis in mathematics defended in July 2004.

Manuscript received August 22, 2004; revised October 26, 2004

Each element of the sequence arises by summing up the two elements that come before it. We have put the first twelve Fibonacci numbers into a table.

n	0	1	2	3	4	5	6	7	8	9	10	11	12
$Fib(n)$	0	1	1	2	3	5	8	13	21	34	55	89	144

Although this definition is formulated very simply in natural language it caused some troubles during its formalization in [1]. Since [1] was written with the primary aim of coding some algorithm in the Mizar model of a computer (AMI), and the expressive power of the Mizar language in those days was significantly weaker than it is today, nobody complained.

```
definition let n be Nat;
  func Fib (n) -> Nat means
:: PRE_FF:def 1
  ex fib being Function of NAT, [:NAT, NAT:] st
   it = (fib.n)`1 & fib.0 = [0,1] &
   for n being Nat holds fib.(n+1) = [ (fib.n)`2, (fib.n)`1 + (fib.n)`2 ];
end;
```

In the next section we will propose a slight modification of this definition based on schemes for recursive definitions available in the Mizar Mathematical Library to simplify the definition.

3. Recursive Definitions in Mizar

In order to make the definition clearer and closer to mathematical standards, first we define a function which maps the set of all natural numbers into the set of corresponding Fibonacci numbers.

```
definition
  func Fib -> Function of NAT, NAT means
    it.0 = 0 &
    it.1 = 1 &
    for k being Nat holds it.(k+2) = it.(k+1) + it.k;
end;
```

Then we define a Mizar functor which is equivalent to the original definition. To prove the correctness of a functor definition in Mizar one have to show **existence** (which is often just the construction of the object with appropriate properties) and **uniqueness** (the proof that such an object is unique) conditions. In the case of Fibonacci-type sequences, both can be justified easily by the schemes introduced recently in [5].

The scheme of existence is as follows:

```
scheme LambdaRec2ExD { X() -> non empty set,
                       A, B() -> Element of X(),
                       F(set, set, set) -> Element of X() } :
  ex f being Function of NAT, X() st
    f.0 = A() &
    f.1 = B() &
    for n being Nat holds f.(n+2) = F(n, f.n, f.(n+1));
```

A and B here are starting values of a recursively defined sequence. One has to use a private functor which defines a recursive step. Similarly the scheme of uniqueness – **LambdaRec2UnD** – is also proven. These schemes enable us to have the correctness proof of the definition nearly straightforward.

```
definition
  deffunc F(Nat,Nat,Nat) = $3 + $2;
  func Fib -> Function of NAT, NAT means
    it.0 = 0 & it.1 = 1 &
      for k being Nat holds it.(k+2) = it.(k+1) + it.k;
  existence
  proof
    ex f being Function of NAT,NAT st f.0 = 0 & f.1 = 1 &
      for n being Nat holds f.(n+2) = F(n, f.n, f.(n+1))
        from LambdaRec2ExD;
    hence thesis;
  end;
  uniqueness;
end;
```

Obviously, similar shortcut may be applied in the case of the proof of uniqueness (scheme **LambdaRec2UnD** from [5] should be used).

4. Basic Identities, Golden Ratio

According to the recent policy of the Library Committee of the Association of Mizar Users to improve the clustering of the MML items to separate its classical and abstract parts and to further enlarge of the Encyclopedia of Mathematics in Mizar, we developed in [4] a sort of monograph of Fibonacci numbers gathering the most useful and well-known properties of these objects. We formalized the following:

– the Catalan identity (as more general than that of Cassini):

$$(Fib(n))^2 - Fib(n+r)\cdot Fib(n-r) = (-1)^{n-r}\cdot (Fib(r))^2,$$

– the Cassini identity (proven as a corollary of the previous one):

$$F(n+1)\cdot Fib(n-1) - \big(Fib(n)\big)^2 = (-1)^n,$$

– the d'Ocagne identity:

$$Fib(m)\cdot Fib(n+1) - Fib(n)\cdot Fib(m+1) = (-1)^n\cdot Fib(m-n),$$

– the Gelin – Cesaro identity:

$$(Fib(n))^4 - Fib(n-2)\cdot Fib(n-1)\cdot Fib(n+1)\cdot Fib(n+2) = 1,$$

and the connections with Pythagorean triples in the approach of [7] which will be described in more detail in Section 6.

We have also shown further connections between Fibonacci sequences and the golden ratio factor

$$\tau = \frac{1+\sqrt{5}}{2}$$

as defined in [6].

As the core result, the Binet formula or the theorem stating that the limit of the quotient of two consecutive Fibonacci numbers converges to τ, may be presented [6]:

```
theorem
  for F being Real_Sequence st
    (for n being Nat holds F.n = Fib(n+1)/Fib(n)) holds
      F is convergent & lim F = tau;
```

Most basic equalities which are true for the Fibonacci numbers are also unproblematic to compute for modern computer algebra systems. Because the implementation of computer algebra in the Mizar system is at a very early stage and there is no syntax for recursion in the definitions of the Mizar functors, Mizar encoding of this topic is not straightforward.

At the end of this section we would like to mention one of the advantages of using functions instead of Mizar functors. Namely, we can define various restrictions of an original function **Fib** (infinite as a rule) to obtain via redefinitions and clusters mechanism proper types to use a functor **Sum**.[1] In this manner, we prove two theorems about the summing of Fibonacci numbers with odd and even indices.

One of them is as follows:

```
theorem :: FIB_NUM2:67
  for n being Nat holds
    Sum EvenFibs (2 * n + 2) = Fib (2 * n + 3) - 1;
```

where **EvenFibs(n)** denotes a finite sequence of all Fibonacci numbers with even indices less than or equal to n.

5. The Carmichael Theorem

The Carmichael theorem [2] which was introduced in a paper from 1913 states that if we consider the prime factors of the n-th Fibonacci number (except $n = 1, 2, 6, 12$), then at least one of them has not appeared in the factorization of any earlier Fibonacci numbers (the so-called *characteristic or genome factor*).

The characteristic factors (as well as all prime indices) are in bold face font in Table 1.

This does not mean that for an arbitrary prime natural p the element $Fib(p)$ itself must be prime, only that no smaller Fibonacci number can be a factor. As an example of this, we can point out $Fib(19)$ which is not prime.

```
theorem :: FIB_NUM2:71
  for n being non empty Nat holds
    m is prime & n is prime & m divides Fib (n) implies
      for r being Nat st r < n & r <> 0 holds not m divides Fib (r);
```

The above theorem is based on the observation of Shane Findley only a few years ago (1999, [3]) stating that if we restrict our considerations to Fibonacci numbers generated by primes, *none of them* has appeared a factor in any earlier Fibonacci number (all factors are genome factors).

[1] We should really use sums of partial functions rather than of finite sequences to make this work even easier since we can omit reorderings in this way.

Table 1. Factorization of the first 26 Fibonacci numbers

n	$Fib(n)$	Factors	n	$Fib(n)$	Factors
1	1	1	14	377	$13 \cdot$ **29**
2	1	1	15	610	$2 \cdot 5 \cdot$ **61**
3	2	**2**	16	987	$3 \cdot 7 \cdot$ **47**
4	3	**3**	**17**	1597	**1597**
5	5	**5**	18	2584	$2^3 \cdot 17 \cdot$ **19**
6	8	2^3	**19**	4181	**37** $\cdot$ **113**
7	13	**13**	20	6765	$3 \cdot 5 \cdot 11 \cdot$ **41**
8	21	$3 \cdot$ **7**	21	10946	$2 \cdot 13 \cdot$ **421**
9	34	$2 \cdot$ **17**	22	17711	$89 \cdot$ **199**
10	55	$5 \cdot$ **11**	**23**	28657	**28657**
11	89	**89**	24	46368	$2^5 \cdot 3^2 \cdot 7 \cdot$ **23**
12	144	$2^4 \cdot 3^2$	25	75025	$5^2 \cdot$ **3001**
13	233	**233**	26	121393	$233 \cdot$ **521**

As the proof of the full Carmichael theorem is rather long (even its simplified version in [9] is six pages long), we developed full Mizar formalization of the restricted version with its rather elementary proof given from scratch.

To prove this version of the Carmichael Theorem in Mizar we had to show that for any non-zero prime natural n and a prime natural number m such that m divides $Fib(n)$ we have that if k is non-zero and $k < n$, then m does not divide $Fib(k)$.

The proof is by contradiction. Namely, we assume that there exists a non-zero natural number $k < n$ such that m divides $Fib(k)$, then we obtain that k successively becomes less than $n/4, n/5, n/6$ etc. So there is no smallest $r > 0$.

```
    defpred P[Nat] means
      (m divides Fib (n -' k * ($1 + 1)) & k <= n / ($1 + 2));
A9: P[0];
A14:for s being Nat st P[s] holds P[s+1];
A34:for s being Nat holds P[s] from NAT_1:sch 1(A9,A14);
```

In the above proof skeleton, one may see an appropriate private predicate to use with the scheme of induction, **A9** – the first step of induction and **A14** – the second step. The whole Mizar proof of this theorem consists of 130 lines of code.

Before I present the sketch of the proof, I will cite some lemmas which will be necessary later. The first of them is **FIB_NUM2:42** which states:

```
  for k being non empty Nat holds
    Fib (n+k) = Fib (k) * Fib (n+1) + Fib (k-'1) * Fib (n);
```

Next is theorem Th69 from the same file:

```
  for n being Nat holds Fib (n), Fib (n+1) are_relative_prime;
```

Let us recall that there is at least one natural number, let us call it k, such that $k \neq 0$ and $k < n$ and m divides $Fib(k)$. Assume that k is the smallest natural (positive) number satisfying the above conditions. Then by theorem **FIB_NUM2:42** we can show that

$$Fib(n) = Fib(n-k+k) = Fib(n-k-1) \cdot Fib(k) + Fib(n-k) \cdot Fib(k+1).$$

From the assumption we know that m divides $Fib(k)$. Hence by **NAT_1:56** we have that m divides $Fib(n-k-1)\cdot Fib(k)$. We also know that m divides $Fib(n)$ and this in connection with the fact above gives us that m has to divide

$$Fib(n-k)\cdot Fib(k+1)$$

(by **NAT_1:57**). But according to the theorem **FIB_NUM2:69** both $Fib(k)$ and $Fib(k+1)$ are relative prime which gives us that m does not divide $Fib(k+1)$, so it means that m divides $Fib(n-k)$. The condition $n-k\geq k$ must be satisfied because k is the smallest natural number divisible by m. Hence, we have that

$$k\leq \frac{n}{2}$$

which completes the proof of **A9**.

So, let us assume that the fact above is true for some s and show that it is also true for $s+1$. Let

$$m|Fib\big(n-k(s+1)\big).$$

By theorem Th42 from **FIB_NUM2** we have:

$$Fib\big(n-k(s+1)\big)=Fib\big(n-k(s+2)+k\big)=$$

$$=Fib\big(n-k(s+2)-1\big)\cdot Fib(k)+Fib\big(n-k(s+2)\big)Fib(k+1).$$

From the main assumption $m|Fib(k)$ and theorem **NAT_1:56** we know that $m|Fib\big(n-k(s+2)-1\big)\cdot Fib(k)$. So, we also obtain that

$$m|Fib\big(n-k(s+2)\big)\cdot Fib(k+1)$$

(by **NAT_1:57** and inductional assumption). But m does not divide $Fib(k+1)$ and hence $m|Fib(n-k(s+2))$. As k is the smallest natural number such that $m|Fib(k)$, we then get $n-k(s+2)\geq k$ and finally $k\leq \frac{n}{s+3}$ which completes this part of our proof (**A14**).

So by the scheme of induction, we know that for all natural numbers the following inequality holds:

$$k\leq \frac{n}{s+2}.$$

But, if s is arbitrarily large, then zero is the limit of $\frac{n}{s+2}$ and it means that $k\leq 0$, which is contradictory with the assumption saying k is a natural number not equal to zero. This contradiction completes the whole proof.

6. Pythagorean Triples

We studied also the connections between Fibonacci numbers and Pythagorean triples. We can generate these triples using the following algorithm:

1. Multiply the two inner numbers, then double the result. This is one side of the Pythagorean triangle.
2. To obtain the second side we multiply together the two outer numbers.
3. The third side, the longest one, is found by adding together the squares of the inner two numbers.

The formal definition of the triples is a definition of a Mizar mode.

```
definition
  mode Pythagorean_triple -> Subset of NAT means
:: PYTHTRIP:def 4
  ex a,b,c being Nat st a^2 + b^2 = c^2 & it = { a,b,c };
end;
```

Here one may also see a result of a formalized algorithm from [4].

```
theorem
  for n being non empty Nat holds
    { Fib (n) * Fib (n+3),
      2 * Fib (n+1) * Fib (n+2),
      (Fib (n+1)) ^2 + (Fib (n+2)) ^2 } is Pythagorean_triple;
```

7. Generalizations

As one may expect, the Fibonacci sequence itself is also a subject for generalizations and modifications.

We can point out here an extension of (1) for integer numbers, given by a following formula:

$$Fib(-n) = (-1)^{n-1} \cdot Fib(n).$$

We put the first ten Fibonacci numbers computed according to the above formula into the table below.

n	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
$F(n)$	1	-1	2	-3	5	-8	13	-21	34	-55

The problem with this formal approach is that there is no convenient way of extending conservatively (e.g., for integers) the definition which is introduced already (for natural numbers in our case). The best idea probably is to revise the original, but unfortunately this work is not done.

Another generalization of the Fibonacci sequence we could mention here are Lucas sequences and so-called G–numbers. Obviously, they are based on a recursive relationship similar as (1). The table below summarizes basic differences between them.

Sequence type	1^{st} value $(n = 0)$	2^{nd} value $(n = 1)$
Fibonacci	0	1
Lucas	2	1
G-Fib	a	b

Both ways of generalizations as well as properties analogous to those proved in [6] and [4] are fully formalized in [8]. For example, one may find there a theorem confirming that by using G–numbers we can define Fibonacci and Lucas numbers (Thms. 37, 38, 42).

$$G\text{-}Fib(0,1,n) = Fib(n)$$
$$G\text{-}Fib(2,1,n) = Lucas(n)$$
$$G\text{-}Fib(a,b,n) = a \cdot Fib(n-1) + b \cdot Fib(n)$$

One of the most significant theorems of [8] is the Binet formula for generalized Fibonacci numbers:

```
theorem :: FIB_NUM3:54
  for a, b, n being Nat holds
    GenFib(a,b,n) = ((a * (-tau_bar) + b) * tau to_power n +
                    (a * tau - b) * tau_bar to_power n) / sqrt(5);
```

Interestingly, the proofs of the Binet formula for Fibonacci numbers and generalized Fibonacci numbers have the same length measured in lines of Mizar source code (namely 75). The Mizar functor **tau_bar** corresponds to $\bar{\tau}$ and is equal to $1 - \tau$.

8. Conclusion and Future Work

Our main plan for future is codifying the proof of the (full) Carmichael Theorem (according to Yabuta [9]). Another subject is thinking about generalized Fibonacci numbers (with arbitrary starting values) as a primitive notion with properties of Fibonacci numbers treated as corollaries. We would also like to find out which of the following approaches is more feasible:

- using a function, then to define a functor or
- having Mizar functors as primitive notions, then functions,

because, handling both in parallel seems to be ineffective either during the writing of theorems or during their retrieve from the database.

It is hardy acceptable to have many parallel notions, e.g., squares (various powers, including **|^2**, **^2**, **to_power 2**), sums of series (**Sum**, **Partial_Sums**), etc.

Making coherent sets of theorems written in a uniform style makes some sense, as well as grouping lemmas gathered across the whole MML. It is not clear if dividing the library into abstract and concrete parts will ever succeed, but to uncover information which is deeeply hidden between the lines seems to be the proper way of thinking for the next 15 years of MML.

References

1. G. Bancerek and P. Rudnicki, *Two programs for SCM. Part I – preliminaries,* Formalized Mathematics, 4(1), 1993, pp. 69–72, MML Id: **PRE_FF**.
2. R.D. Carmichael, *On the numerical factors of the arithmetic forms* $\alpha^n + \beta^n$, Annals of Mathematics, 15, 1913, pp. 30–70.
3. S. Findley, unpublished note, see **http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/whatsnew19992002.html**.
4. M. Jastrzębska and A. Grabowski, *Some Properties of Fibonacci Numbers,* to appear in Formalized Mathematics, 12(3), 2004, MML Id: **FIB_NUM2**.
5. A. Korniłowicz, Recursive definitions. Part II, Formalized Mathematics, 12(2), 2004, pp. 167–172, MML Id: **RECDEF_2**.
6. R.M. Solovay, *Fibonacci numbers,* Formalized Mathematics, 10(2), 2002, pp. 81–83, MML Id: **FIB_NUM**.
7. F. Wiedijk, *Pythagorean triples,* Formalized Mathematics, 9(4), 2001, pp. 809–812, MML Id: **PYTHTRIP**.
8. P. Wojtecki and A. Grabowski, *Lucas numbers and generalized Fibonacci numbers,* to appear in Formalized Mathematics, 12(3), 2004, MML Id: **FIB_NUM3**.
9. M. Yabuta, *A simple proof of Carmichael's theorem of primitive divisors,* The Fibonacci Quarterly, 39(5), 2001, pp. 439–443.

MECHANIZED MATHEMATICS AND ITS APPLICATIONS, VOL. 4, NO. 1, MARCH 2005, 83:88

Alcor: A user interface for Mizar

Paul Cairns

UCL Interaction Centre
University College London
London WC1E 7DP, UK
p.cairns@ucl.ac.uk

Abstract – The Alcor user interface to the Mizar library is intended to provide a test bed for exploring how a mathematician might interact with mathematical knowledge management tools. Specifically, how can a mathematician whilst working on or writing up mathematics look up relevant mathematical knowledge without interrupting their workflow? We describe how a specific interaction style has been used to implement keyword search and discuss how that style could benefit other more complex forms of context-specific searching.

1. Interfaces to MKM

The major goal of mathematical knowledge management (MKM) is to make available the enormous volume of mathematical knowledge available in current mathematical resources [9]. Of course, to make that knowledge valuable it needs to be provided to the right person, at the right time, *in the right way*. The potential user-base for MKM is almost anybody who applies or develops mathematical knowledge as part of the their work, that is, mathematicians, physicists, economists, chemists, social scientists and so on. It is therefore unlikely to be able to provide a single user-interface that would be acceptable to all users. The focus then in the current work is to look at user interfaces that might support mathematicians in developing mathematical material that could then integrate back in to MKM resources for other mathematicians. Even so, it is expected that mathematicians could produce and draw upon a wide-range of heterogeneous resources such as web-sites, journal articles, software and text-processing documents such as LaTeX.

This general task of integrating knowledge across distributed, heterogeneous resources with multiple authors is a typical knowledge management task[8]. However, it is further complicated in the case of mathematics that any synthesis of this material that a mathematician might make and build upon needs to be re-integrated back into the body of mathematical knowledge. There is no room for imprecision or personal interpretation of meaning as this could mean the difference between a correct proof and an incorrect one.

Mathematical knowledge management is still some way from achieving such an integrated environment for the working mathematician. I accordingly do not try to address the full problem but instead consider a mathematician who might be working using some homogeneous library of mathematics and who would like in turn to contribute their work back to the library. The prototype environment would be an author for the Mizar Mathematical Library [6]. To be a successful Mizar author, a person must add an article to the library that constitutes some significant mathematical contribution to the library and in addition it should avoid replicating or apparently contradicting the definitions and theorems already in the library. To explore the working activities of such an author, the Alcor system has been developed as a possible working environment.

Manuscript received September 1, 2004; revised September 20, 2004

This paper describes the main motivations for developing the Alcor system. The current system falls considerably short of these lofty goals, nonetheless, we describe its current features and capabilities before concluding with the planned future developments.

1.1 A note on the name

The Mizar system is a proof checking environment built on a sophisticated mathematical vernacular. It is famously named after the second star (in fact, quadruple star system) in the handle of the constellation *Ursa Major*, also designated ζ UMa. Interestingly, Mizar has a close companion star, Alcor, not in the main Mizar system. Alcor is visible to the naked eye for those with good acuity. Native Americans actually called the pair of stars the horse and rider [7]. Inasmuch as the system described here proposes to sit alongside the Mizar system rather than augment or integrate into it, the name Alcor seemed appropriate in many ways.

2. Using Mizar

Though the motivation in developing Alcor was to explore how a working mathematician might use Mizar or a system like Mizar, a user-centred design approach that would be typical for user interface development [11] has not been adopted. This is for the simple reason that there are currently very few working mathematicians who regularly use any sort of automated proving or proof checking system. The proposed developments in this system and in mathematical knowledge management more generally are innovative, fitting into a complex working pattern [8] and so a study of existing practices would, at this stage at least, be uninformative. Instead, a problem statement is used to encapsulate the goals of the system whilst not forgetting the priority of the user [10]:

> Design a graphical user interface to enable mathematicians in their ordinary work to successfully develop Mizar articles that do not replicate existing Mizar content and without the need for extensive knowledge of the entire Mizar library.

The approach to addressing this problem statement is motivated by a personal view of the difficulties in developing a Mizar article. A full discussion of Mizar and the process of developing articles would be inappropriate here and additionally it has been discussed in great detail elsewhere [12, 15]. In brief, though, a Mizar article is a collection of theorems, definitions and proofs written in the Mizar language. The language is very rich, particularly in comparison to other formal mathematical languages [14], being intended as a mathematical vernacular though this language is not without its complexities [4]. Having written theorems and proofs in the Mizar language, the Mizar system itself checks the proof and is able to assert whether or not the proof is formally correct within the definition of the language, the context of the particular article and any elements of the library referenced in the article. The goal, then, of any author is to produce theorems that are significant mathematically but not previously present in the library.

There are some natural barriers to achieving this goal. The library is very large with some 2000 definitions and 30,000 theorems. Full knowledge of it may only come with years of experience. In addition, like all formal mathematical languages there is a degree of verbosity [14] which means that not all theorems are of mathematical significance. Finding the key theorems for a particular task could be difficult. Also, like all formal languages, such as programming languages, it can sometimes be easier at first to copy and then adapt existing proofs. Even then, it is important to know that the terms in the proofs are being used according to their correct definitions.

Given this goal and these possible barriers, a person wishing to write a new article may well ask the following questions:

1. What is already in the library that I might need?
2. Can I adapt a proof of a similar statement?
3. Is there a theorem that I could use to prove this. . . ?
4. What is the exact definition of. . . ?

These first two questions work at a very high level being very much about the topic of work. The last two questions are more about the pragmatics of producing a correct Mizar proof. In particular, I am also working on the third question using LSI to provide a new search technique that may eventually address questions 1 and 2 [3]. The last question is relatively straightforward and it is this that is directly addressed in the current version of the Alcor system. It is hoped that as work on LSI progresses, Alcor will increasingly support answering the other three questions.

The need for better search facilities is recognised by frequent users (and developers) of Mizar [2], indeed, Bancerek and Rudnicki ask similar questions when it comes to searching Mizar. Their approach has been to develop a query language to do semantic retrieval that addresses the last two of the above questions. With the Alcor system, the approach has been rather to take a more graphical "point-and-click" style though at the moment this is at the cost of being less powerful in the kinds of retrieval that it can do.

3. The Alcor System

Given the current goal of the Alcor system is to be able to find definitions for an author working on an article, a workflow approach to authoring was taken inspired by the Phrasier system [5]. More explicitly, the author should have a full text editor that allows them to produce an article but at the same time be integrated with a search tool that enables them to look up definitions without interrupting their attention from their own work.

Figure 1 shows the main screen of the Alcor system. The left hand window is the main editor where the author would be working. There are many ways to enable a simple keyword search for definitions. Several have been included. There is a text box along the bottom of the screen where users can type free text. This text box also has a drop down menu that shows past search terms. This acts like a history function so that the author can easily replicate previously performed searches and peruse them if they have forgotten the exact terms of the search. The third method for entering search terms is simply to highlight words in the editor window. In this way, the author can maintain their focus at the position in their article where they are currently working. Having highlighted the word, they can then click the "Search" button that runs beneath the menu bar (though this in fact is not easily recognised in its current location). Alternatively, in keeping with the PC paradigm, the user could right click and select to search on the highlighted word.

Search results appear in the lower window. Currently the display results are the location, that is, the Mizar reference, for the found items and also the type of item that has been found. This is because a single term may be defined in multiple senses depending on the particular need in particular articles. For example, the figure shows the results of a search on the term "closed" that has resulted in 9 items found. The context of work is an article on topology but of course with a novel piece of mathematics, it could be misleading to assume that only the topological definition of closed is required. For example, if the article were on topological groups then closed could be needed in either the sense for topologically closed or for algebraically closed.

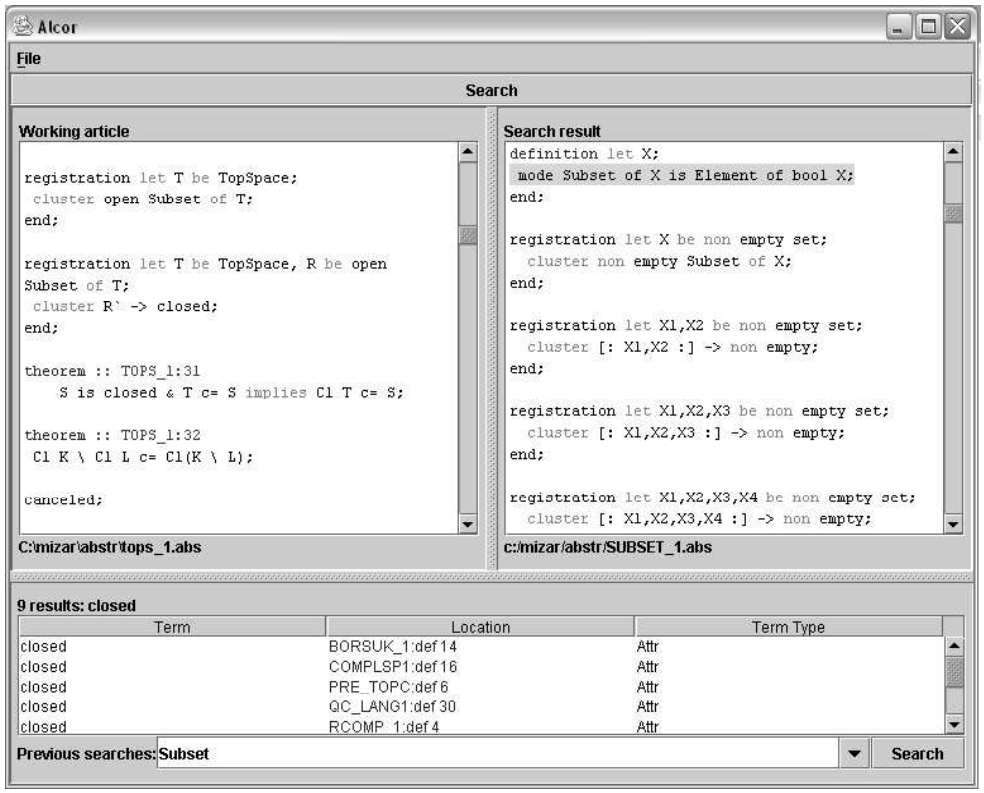

Fig. 1. The Alcor system

To see the actual definitions, the user simply clicks on the location in the search results window. The required definition is displayed in the right hand window as given in the abstract (rather than the full Mizar article). This is partly simply to demonstrate the principle and partly because of implementation limitations in finding the definition in the full article. To make the definition clear to the user, it is positioned at the top of the window and highlighted in yellow.

As is often the case, the results of one search may lead on to further searches to clarify or explore other possibilities. The right hand window therefore also supports the same interactions for performing searches. Thus, the author could highlight search terms in their own work, get a retrieved abstract in the other window and carry on highlighting new search terms in the retrieved abstract. New retrieved abstracts would still come up in the right hand window so that the author would always have their own work on display.

Thus, a keyword search for definitions is implemented in a traditional graphical user interface style. In this sense, it is very like the Phrasier system where a text document being written, say a newspaper article, acts as a context for searching for similar or related articles. The found articles are shown in parallel to the article being worked on. In a natural language context, this could act as the “plagiarist’s dream” (I. Witten, personal communication, 2002) but for mathematics where there is much less profit in plagiarism this workflow may be of real benefit.

Indeed, it is hoped that this has a more integrated workflow than the use of either a separate query language or a separate tool entirely such as **`grep`**. Also, the style of interaction is clearly extensible to other sorts of search. For example, to find a theorem that might help in proving a particular statement, the author could highlight the statement to be proved (rather

than just a single word) and search on that. And highlighting a theorem might search for related theorems whose proofs could be adapted to prove the author's statement. In all of these examples, the context of the current work, namely the article that the author is currently working on, indicates the kind of query that is required. This removes the onus from the author to formulate the query precisely and so allows them to concentrate only on authoring.

Of course, achieving this context-aware searching requires appropriate underlying algorithms. This is the aim of other work [3] but it is worth noting that the interaction style is actually independent of the methods use to produce good search results.

4. Future Work

In its current state, the Alcor system is intended as a test bed for different types of interaction that might support an author. In particular, the initial focus has been on supporting searching the Mizar Library. Thus, Alcor is far from a complete authoring environment for Mizar articles — it certainly does not attempt to offer the full functionality that the Mizar mode in emacs offers. In fact, so far, it only colours keywords in approximately the same way as the emacs Mizar mode. A first step before trying out Alcor with real Mizar authors would be to augment it to be at least as good as the emacs mode with the additional interactions discussed here. That way, authors would not be disadvantaged over using existing tools and and investigation of the real efficacy of the proposed interactions could be done.

The fact that searches produce retrieved articles that can then be used to define further searches means that there is the possibility of an author losing useful searches whilst they explore other possibilities. There is already a history mechanism for search terms but a more sophisticated history mechanism including forward and back as in a web browser and found items listed by article may also be required.

Alcor is intended to support *novel* search methods for the Mizar library and my other work is looking at how to implement that. A specific problem with the LSI method is how to formulate suitable queries so it is hoped that Alcor would provide sufficient support to automatically formulate queries based on the context of searching. In fact, the highlighting style of defining search terms could give valuable information as to the exact nature of the search being done. In this sense, the style of interaction would become integral to the search algorithm. This may not be desirable in the long run and other methods of defining context may be explored in future versions of Alcor.

References

1. Asperti, A., Buchberger, B., Davenport, J. H.: Mathematical Knowledge Management, Proceedings of MKM 2003. LNCS **2594** Springer Verlag (2003)
2. Bancerek, G., Rudnicki, P.: Information Retrieval in MML. In [1] (2003) 119–132
3. Cairns, P.: Informalising Formal Mathematics. To appear in MKM 2004, 3rd Int. Conf on Mathematical Knowledge Management (2004)
4. Cairns, P., Gow, J.: Using and Parsing the Mizar Language. Electronic Notes in Theoretical Computer Science **93** Elsevier (2004) 60–69
5. Jones, S., Stavely, M. S.: Phrasier: a system for interactive document retrieval using keyphrases. *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval* (1999) 160–167
6. Journal of Formalised Mathematics
   `http://mizar.org/JFM`
7. Menzel, D.: A Field Guide to the Stars and Planets. Collins (1964)
8. Mirel, B.:Interaction Design for Complex Problem Solving. Morgan Kaufmann (2004)

9. MKM 04 web-site, accessed 9th August, 2004
http://www.mizar.org/MKM2004/
10. Newman, W. M., Lamming, M. G.: Interactive System Design. Addison-Wesley (1995)
11. Preece, J., Sharp, H. and Rogers, Y.: Interaction Design. Wiley (2002)
12. Rudnicki, P.: An overview of the Mizar project. In Proceedings of 1992 Workshop on Types and Proofs for Programs (1992)
13. Wiedijk F.: The DeBruijn Factor. Note accessed 9th August, 2004
http://www.cs.kun.nl/ freek/factor/factor.pdf
14. Wiedijk F.: Comparing Mathematical Provers. In [1] (2003) 188–202
15. Wiedijk F.: Mizar: An Impression. Note accessed 9th August, 2004
http://www.cs.kun.nl/ freek/mizar/mizarintro.pdf

MECHANIZED MATHEMATICS AND ITS APPLICATIONS, VOL. 4, NO. 1, MARCH 2005, 89:94

e-Learning and Mizar

Yasunari Shidama

Shinshu University, Faculty of Engineering
Dept. of Information Engineering
380-8553 Nagano-ken Nagano-shi Wakasato 4-17-1
shidama@cs.shinshu-u.ac.jp

1. Internet and e-Learning

With the development of Internet and information communication technologies, by simply connecting their PCs to the Internet, students today can study college lectures at any time and any place without going to campus. In colleges and universities in the U.S., where persons with jobs in the working force comprise half of the student population, e-learning technologies were introduced first in the world using Web contents and various information communication technologies such as e-mail and BBS or electronic bulletin board systems. The University of Phoenix e-learning program and the OpenCourseWare project of MIT are well-known examples.

In 2002, the department of information engineering at Shinshu University established the first Internet-based graduate school in Japan which allows students to earn degrees at the master and Ph.D. levels through distance courses and research advising via the Internet. The URL of the Internet graduate school is **http://cai.cs.shinshu-u.ac.jp/sugsi/** and it is the entrance point from which students are able to access the research and education services of the school which include lectures and lecture materials, online exams, research advising, online discussions with professors, seminar participation, etc.

Discussions of e-learning have reached a point where web materials and contents focusing on visual effects have lost their novelty. However, these are but a single method of delivery for current e-learning and distance education by the Internet. What we need to focus on is an important topic in the background of e-learning - the emerging shift from group education in the classroom to the individualization of education.

The ultimate form of individualized education would be to provide for each student a one-on-one interaction with one professor (or perhaps a group of professors who are experts in the area).

By escaping the limitations of time and space using the Internet, this type of individualized education becomes realistically possible. In this case, rather than simply a role as an educator, the professor plays more of a role as a coach who assists individual students pursuing learning through the materials on the Internet.

2. Mathematical education using Mizar

It is frequently pointed out that although e-learning is effective as a means for communicating knowledge and information, it is not well-suited for exercises in thinking processes and problem solving. As a result, it is often said that science and engineering education, which are based on logical reasoning, cannot be made into e-learning.

Manuscript received September 17, 2004; revised September 18, 2004

Certainly, in some exercises it is necessary to handle actual materials and equipment and it is not just a matter of replacing them with corresponding items in virtual reality. By no means is e-learning a cure-all remedy. However, if we limit our considerations to the training of logical thinking processes and skills for mathematical thinking, the situation is completely different. For example, in order to teach students to fully understand the proof of even a basic mathematical theorem at a first or second year level of general college education, we will need to either work with very small groups or spend long hours on thorough individualized teaching and guidance based on the progress of each student. This would call for tremendous patience on the part of the professor.

In order to develop students with a talent for creating and advancing scientific technologies, the education and training for logical reasoning and mathematical thinking is essential. Ironically, the richer a country becomes economically, the more there is a tendency for its students to dislike engaging in such logical reasoning and mathematical thinking training which is plain and boring to them.

This is a serious situation which can be found in almost all of the developed countries of the free economy. Japan as well is no exception and this problem is becoming more and more serious. The roots of the “technology nation” which Japan had held pride in until now are starting to see their foundations sink from this “abandoning of the technical fields”. Rather than continuing the development efforts of many predecessors who built and nurtured the scientific and industrial technologies and wanting to contribute to the world, the focus of more and more people is only on economic profits and how to reap the benefits of the full and abundant lifestyles and environment created by the fruits of these developments and this trend .is painting a bleak future for the nation.

The key to recovering from this situation will be nothing other than the revival of mathematics education. However, realistically, as mentioned earlier, we must consider how to secure professors for very small groups of students and again are such lectures which demand long hours of patience from the professors possible?

For this problem, there is no simple solution such as just assigning study exercises for repeatedly solving calculation problems. When we consider, for example, the number of hours of effort that will continue in order to coach and train a student how to construct the answer to a proving problem, we realize that we must employ the power of computers with their never tiring endurance to support the work of training mathematical thinking skills in students.

Researchers in our group have been involved in the Mizar project which develops a library of mathematical theorems using a specification language developed by Polish mathematician Prof. Andrzej Trybulec for the formalization of mathematics. The following will be a brief description for those new to Mizar.

2.1 Mizar

The formal mathematics specification language Mizar is used to write proofs of mathematical theorems using symbolic logic based on first-order predicate logic and a set of axioms of Tarski-Grothendieck set theory. The Mizar system is what is called a proof checker and it employs a computer to verify the logical correctness of the proof of a mathematical theorem specified in the Mizar language. An example of such a proof is shown in below:

```
environ

vocabulary ARYTM_3, ARYTM_1,  INT_1;
notation XCMPLX_0, REAL_1, NAT_1, INT_1;
```

```
constructors REAL_1, NAT_1, INT_1;
clusters INT_1;
requirements  SUBSET, NUMERALS, ARITHM;
theorems INT_1, XCMPLX_1;
schemes NAT_1;

begin

theorem
  for n being Nat holds 2 divides n*(n-1)
  proof
    defpred P[Nat] means 2 divides $1*($1-1);
    0 = 2 * 0;
    then
P0: P[0] by INT_1:def 9;
PN: for n being Nat st P[n] holds P[n+1]
    proof
      let n be Nat;
      assume P[n];
      then consider s being Integer such that
A2:   n * (n - 1) = 2 * s by INT_1:def 9;
      (n+1)*(n+1-1) = n*(n-1) + n + n*1
                    .= 2*(s+n) by A2;
      hence P[n+1] by INT_1:def 9;
    end;
    thus for n being Nat holds P[n] from NAT_1:sch 1(P0,PN);
  end;
```

Here, **INT_1** is a library of definitions and theorems already registered in the Mizar system. The definition **INT_1:def 9** is specified as shown below:

```
definition
  let i1,i2 be Integer;
  pred i1 divides i2 means
:: INT_1:def 9
    ex i3 st i2 = i1 * i3;
  reflexivity;
end;
```

Similarly, **NAT_1:sch 1** represents the mathematical scheme of induction registered as follows:

```
scheme Ind { P[Nat] } :
  for k being Nat holds P[k]
provided
  P[0] and
  for k being Nat st P[k] holds P[k + 1];
```

Details of the Mizar grammar can be found in such literature as the documentation by Prof. Yatsuka Nakamura of Shinshu University which is available at the following URL

http://markun.cs.shinshu-u.ac.jp/kiso/projects/proofchecker/mizar/Mizar-E/Miz-etit.htm

By examining the lines after **begin**, researchers with some background information will be able to make out in most part that if Nat represents the natural numbers, this is a specification for the proposition, "For any natural number n, $n \cdot (n-1)$ is divisable by 2" and its proof by induction.

A special characteristic of the Mizar proof-checker is that specifications can be made with expressions close to natural language. There are a number of proof-checkers of this type in the world, but there are many cases in which the development was done by a graduate student for thesis work and once the student graduates no one else uses it again. The Mizar proof-checker, however, is developed by the Mizar group headed by Polish mathematician Prof. Andrzej Trybulec. A great number of researchers and graduate students work together in maintaining and expanding the library. Presently, approximately 800 mathematical articles are contained in the library and both the Mizar system and library are completely open at

http://mizar.uwb.edu.pl/

A mirror site in Japan is available at:

http://markun.cs.shinshu-u.ac.jp/mirror/mizar/

2.2 Mizar system on the web

Prof. Yatsuka Nakamura and I represent the members of the Mizar group on the Japan side and currently manage the Internet graduate school program at Shinshu University. Mizar holds an important position in our online course materials and we have two Mizar systems on the Web for students to use.

One is the megrez system, developed by Prof. Grzegorz Bancerek during his JSPS fellowship stay in Japan available at

http://megrez.mizar.org/

The other was created by a Japanese professor and can be accessed at the address

http://www.wakasato.org/mizar/

The latter is mainly used for graduate student lecture courses. To explain briefly how it is used, students will first select a package depending on whether they want to define new terms or use special system commands. In the next screen, they will input their student ID numbers and e-mail addresses and be guided to the main page of the verifier. Here, students may input proofs written in the Mizar language to a textbox field or specify a prepared file containing the proof to be checked. Clicking the SUBMIT button with a mouse will start the Mizar proof checker and any errors detected in the proofs will be displayed on the screen. With this system, students will write proofs, in the same manner as programming exercises, ranging in difficulty from simple propositions like the example given earlier to proofs of theorems in differential and integral calculus of first and second year general education courses as shown here:

```
theorem
  for n be Nat, f be PartFunc of REAL,REAL, x0,r be Real
    st ( 0 < r & f is_differentiable_on n+1, ].x0-r,x0+r.[ )
    for x be Real st x in ].x0-r, x0+r.[ holds
```

```
ex s be Real st 0 < s & s < 1 &
f.x=Partial_Sums(Taylor(f, ].x0-r,x0+r.[,x0,x)).n
+ (diff(f,].x0-r,x0+r.[).(n+1)).(x0+s*(x-x0))
* (x-x0) |^ (n+1) /((n+1)!);
```

Unlike in programming exercises, however, as the reader familiar with symbolic logic can easily anticipate, attempts to specify the proof of a fairly long mathematical theorem in a haphazard hit-or-miss fashion without careful planning will usually end in failure and frustration. From the start, proofs need to be mapped out systematically with appropriate subproofs and lemmas placed at intermediate steps and throughout the development work, a step-by-step application of logical thought processes will be indispensable for completing the proof.

Since we are dealing with the specification of a proof, there can be more than one "correct answer". Using the available axioms and other theorems which have been proved, one needs only to prove the conclusion logically so we can easily have as many possible answers as there are students.

For example, if we consider even the simple example above, there are a number of ways of transforming n*(n-1) into 2*(s+n). It is obvious then that if the assignment is a fairly difficult proof problem, there will be a number of approaches for simply planning the proof and we can easily find ourselves in a situation where each student has a different and correct proof solution. Attempting to grade such kinds of exercises in a report format will require a professor to spend a considerable amount of time reading and checking each proof variation. Furthermore, before all students reach the goal of building a correct proof, these reports will be read, corrected, and returned for resubmission a number of times at the cost of extraordinary amounts of time and energy from a professor.

With proof-checker systems, we have simultaneous use by as many students as is allowed by the performance and power of our computers. The proof-checker can also work on the individual PCs of each student. The proof-checker never gets tired or emotional and is able to mechanically check and point out errors millions of times on end until a student has finally built a correct proof.

3. Construction plans for the Mizar library

The continued active development of the Mizar library is made possible through the efforts of the Mizar group. Japanese university researchers are also involved in this project. Students of the Internet graduate school described earlier also participate in this project. Prof. Yatsuka Nakamura is involved in writing a group of library articles for the Jordan curve theory and we also work in collaboration on a group of articles for functional analysis. The objective of the portion of library articles I am involved with is to formalize by 2010 all of the fundamental theorems of differential and integral calculus and functional analysis that would be studied by undergraduate and graduate students. The areas include differentiation and integration of single- and multivariable real and imaginary functions, Lebesgue integration, topological linear spaces, Hilbert spaces, Banach spaces, Lp-spaces, Sobolev spaces, theory of linear operators, finite and infinite dimension equilibrium point theory, etc. The list continues to grows on, but the development is progressing steadily bit by bit.

Without a doubt, we can expect in the future practically all mathematical theorems to be added to the library and researchers in the areas of mathematics, mathematical sciences, and applied mathematics will be able to write their own research results and the accompanying proofs in Mizar. The correctness of such proofs, no matter how unknown the name of the author is, would be checked not by persons of authority in scientific societies, but instead by instant

computer verifications. The system of human readers would serve to judge the usefulness of scientific results at a higher level as only humans can. Then, a truly ubiquitous environment for academic research in mathematical science fields can be achieved.

Contents