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 ∈, 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 · 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.