Basic Elements of Computer Algebra in MIZAR

Adam Naumowicz[†] *Czeslaw Bylinski*[††]

[†]Institute of Computer Science
University of Bialystok, Poland
adamn@math.uwb.edu.pl

[††]Section of Computer Networks
University of Bialystok, Poland
bylinski@math.uwb.edu.pl

Abstract - In this paper we describe special features of the Mizar system which provide some elements of computer algebra and present how they strengthen the capabilities of the Mizar checker.

Keywords – Mizar checker, computer algebra, requirements directive, properties of Mizar functors and predicates

1. Introduction

The original goal of the Mizar project was to design and implement a software environment that supports writing traditional mathematics papers. Mathematical practice shows that even in formal proofs some easy background reasoning can be reduced. There are many powerful systems that efficiently process numeric and symbolic computation. Similar techniques incorporated into the Mizar system would considerably benefit the Mizar user community. At the moment, the inference checker uses model elimination with stress on processing speed, not power. However, its power can be extended in several ways. In this paper we discuss how properties that can be associated with Mizar definitions and the **requirements** directive can strengthen the process of inference justification in Mizar. Both these features influence how equality classes are generated in the EQUALIZER - the module responsible for the equality calculus in the Mizar checker (cf. [13]). Their effects can substantially reduce the amount of justification an author must provide in a proof. Used in connection with suitable management utilities these features stimulate the growth and evolution of the Mizar Mathematical Library (MML)[1].

2. Properties

As described in [10], there are four main kinds of constructors in Mizar: predicates, functors, modes and attributes. The Mizar system allows for special automated processing of certain properties of the first two types. The properties currently implemented for predicates (constructors of formulae) include: **symmetry**, **asymmetry**, **reflexivity**, **irreflexivity**, and **connectedness**. The properties for functors (constructors of terms) are: **commutativity**, **idempotence**, **involutiveness**, and **projectivity**. When

Manuscript received October 1, 2001; revised July 3, 2002.

[1] The MML is the data base of Mizar articles. The systematic collection started in 1989. At the time of this writing it contains 724 articles (about 54 MB of Mizar texts).

included in a definition of a predicate or a functor, the above-mentioned properties can be automatically used by the Mizar checker in every inference step which concerns that constructor. In that case, corresponding statements and references to these statements become superfluous. The properties are paired with a justification of suitable correctness conditions which we describe below. We also discuss the restrictions which are necessary to avoid a collapse of system consistency.

2.1 Predicate Properties

In general, a Mizar predicate with properties is defined as:

definition
let x be q_1**;**
let y be q_2**;**
pred Example_Pred x,y means
d **(x,y);   ::   the definiens of the predicate**
predicate-property-symbol **proof ... end;**
...
end;

definition
let x be q_1**;**
let y be q_2**;**
pred Example_Pred x,y means
d **(x,y);   ::   the definiens of the predicate**
predicate-property-symbol **proof ... end;**
...
end;

where *predicate-property-symbol* is one of the following: **asymmetry**, **symmetry**, **reflexivity**, **irreflexivity**, and **connectedness**. The properties are accepted only when the types q_1 and q_2 are equal. The following table contains a summary of predicate properties with suitable justification formulae. Examples of all properties taken from MML are presented below.

Predicate property	**Formula to be proved as justification**
asymmetry	for x,y being q_1 holds d (x,y) implies not d (y,x)
symmetry	for x,y being q_1 holds d (x,y) implies d (y,x)
reflexivity	for x being q_1 holds d (x,x)
irreflexivity	for x being q_1 holds not d (x,x)
connectedness	for x,y being q_1 holds not d (x,y) implies d (y,x)

We illustrate **asymmetry** with the Mizar primitive **in** predicate. This predicate has no accompanying justification because it is built into the Mizar system. The article HIDDEN ([6]) documents built-in notions.

definition
 let x,X be set;
  pred x in X;

```
    asymmetry;
  end;
```

As an example of the **symmetry** property, we show a predicate satisfied whenever two sets have an empty intersection (XBOOLE_0:def 7[2], [4]). It sometimes happens, as in this example, that the condition is obvious for the checker and no justification is needed.

```
  definition
   let X,Y be set;
    pred X misses Y means :Def7:
     X /\ Y = {};
    symmetry;
   antonym X meets Y;
  end;
```

An example of **reflexivity** is the divisibility relation for natural numbers (NAT_1:def 3, [1]) presented below:

```
  definition
    let k,l be natural number;
      pred k divides l means :Def3:
        ex t being natural number st l = k * t;
      reflexivity
        proof
          let i be natural number;
          i = i * 1;
          hence thesis;
          end;
  end;
```

An example of a predicate with **irreflexivity** is the proper inclusion of sets (XBOOLE_0:def 8, [4]).

```
  definition let X,Y be set;
   pred X c< Y means :Def8:
    X c= Y & X <> Y;
   irreflexivity;
  end;
```

We demonstrate **connectedness** with the redefinition of inclusion for ordinal numbers (ORDINAL1, [2]).

```
  definition
   let A,B be Ordinal;
   redefine pred A c= B;
     connectedness
      proof
      let A,B be Ordinal;
     A in B or A = B or B in A by Th24;
      hence thesis by Def2;
      end;
```

[2] The phrase *Article-Identifier*:def *Definition-Number* follows the convention which identifies all Mizar definitions in the MML.

```
end;
```

Here, Th24 and Def2 refer to:

```
theorem Th24:
  for A,B being Ordinal holds A in B or A = B or B in A

definition let X be set;
 attr X is epsilon-transitive means :Def2:
  for x being set st x in X holds x c= X;
end;
```

We note that a similar concept could also be implemented for modes since they are in fact special kinds of predicates. For example, **reflexivity** seems useful for a mode constructor like **Subset of**. Also, the set of currently implemented predicate properties is not purely accidental. Since every Mizar predicate can have an antonym, each property has a counterpart related to the antonym. For example, **reflexivity** automatically means **irreflexivity** for an **antonym** and vice versa. The same can be said for the pair **connectedness** and **asymmetry**. Obviously, **symmetry** of an original constructor and its **antonym** are equivalent.

2.2 Functor Properties

The properties of binary functors in Mizar are **commutativity** and **idempotence**. In general, we define a binary functor with properties in the following form:

```
definition
 let x be $q_1$; let y be $q_2$;
  func Example_Func(x,y) -> $q_3$ means
    $d$ (it,x,y);
   binary-functor-property-symbol proof ... end;
   ...
end;
```

where *binary-functor-property-symbol* is **commutativity** or **idempotence**, and the Mizar reserved word 'it' in the definiens denotes the value of the functor being defined.

Binary functor property	**Formula to be proved as justification**
commutativity	for x being q_3, y being q_1, z being q_2 holds d (x,y,z) implies d (x,z,y)
idempotence	for x being q_1 holds d (x,x,x)

An example showing both binary functor properties is the set theoretical join operator (XBOOLE_0:def 2, [4]).

```
definition
 let X,Y be set;
  func X \/ Y -> set means :Def2:
   x in it iff x in X or x in Y;
  existence proof ... end;
  uniqueness proof ... end;
```

```
    commutativity;
    idempotence;
  end;
```

With the current implementation, **commutativity** is only applicable to functors for which the result type is invariant under swapping arguments. Furthermore, **idempotence** requires that the result type be wider than the type of the argument (or equal to it).

The Mizar unary functor with properties uses the form below:

```
  definition
   let x be $q_1$;
    func Example_Func(x) -> $q_2$ means
  $d$ (it,x);
      unary-functor-property-symbol proof ... end;
    ...
  end;
```

where *unary-functor-property-symbol* is **involutiveness** or **projectivity**. The system consistency is protected by the restriction that types q_1 and q_2 be equal.

Unary functor property	**Formula to be proved as justification**
involutiveness	for x,y being q_1 holds d (x,y) implies d (y,x)
projectivity	for x,y being q_1 holds d (x,y) implies d (x,x)

The **involutiveness** property is used with the inverse relation (RELAT_1:def 7, [14]).

```
  definition
   let R be Relation;
    func R~ -> Relation means :Def7:
     [x,y] in it iff [y,x] in R;
    existence  proof ... end;
    uniqueness  proof ... end;
     involutiveness;
  end;
```

As an example of **projectivity** we give the functor for generating the absolute value of a real number (ABSVALUE:def 1, [9]).

```
  definition
   let x be real number;
   func abs x -> real number equals :Def1:
                x if 0 <= x
                  otherwise  -x;
   coherence;
   consistency;
   projectivity by REAL_1:66;
  end;
```

Here, REAL_1:66 ([8]) refers to:

```
  theorem :: REAL_1:66
   for x being real number holds x < 0 iff  0 < -x;
```

Due to some problems in implementation, the **idempotence**, **involutiveness**, and **projectivity** properties are not available for redefined objects as yet.

3. Requirements

The **requirements** directive, which is comparatively new in Mizar[3] allows for special processing of selected constructors. Unlike the properties described in Section 2, it concerns the **environ** part of a Mizar article (cf. [10]). With the **requirements** directive, some built-in concepts for selected constructors will be imported during the accommodation stage of processing an article. In the MML database they are encoded in special files with extension '.dre'. As yet, the special files in use are: HIDDEN, BOOLE, SUBSET, ARYTM, and REAL. We describe how they assist the Mizar checker with the work of reasoning so that the amount of justification an author must provide can be reduced.

3.1 requirements HIDDEN

This directive is automatically included during accommodation of every article and therefore does not need to be used explicitly. It identifies the objects defined in the axiomatic file HIDDEN, i.e., the mode **set** followed by the '**=**' and '**in**' predicates (HIDDEN:def 1 - HIDDEN:def 3, [6]). Mode **set** is the most general Mizar mode and every other mode widens to it. Thanks to the identification provided by **requirements HIDDEN** it is used internally wherever the most basic Mizar type is needed, e.g., while generating various correctness conditions. The fundamental equality predicate '**=**' is extensional which means that two objects of the same kind (atomic formulae, types, functors, attributes) are equal when their arguments are equal. This particular property is used frequently by the Mizar checker. The '**=**' relation is also symmetric, reflexive, and transitive. Predicate '**in**' plays an important role in the "unfolding" of sentences with the Fraenkel operator in positive and negative contexts. This allows sentences of the form **ex y being** *q* **st x=y & P[y]** to be true whenever **x in {y being** *q* **: P[y]}** is true and vice versa. Various features of the '**in**' predicate are considered in conjunction with other requirements (see Sections 3.2, 3.3).

3.2 requirements BOOLE

When processing an article with **requirements BOOLE**, Mizar treats specially the constructors provided for: the empty set (**{}**), attribute **empty**, set theoretical join (\/), meet (/\), difference (\), and symmetric difference (\+\) given in definitions XBOOLE_0:def 1-XBOOLE_0:def 6, [4]). It allows the following frequently used equations to be accepted without any external justification: **X \/ {} = X, X /\ {} = {}, X \ {} = X, {} \ X = {}, and {} \+\ X = X**. The empty set also gets additional properties: **x is empty implies x = {}**, and similarly **x in X implies X is non empty**. Additional features concerning the empty set are also described in the next section[4].

---

[3] Historically, the first **requirements** directive was ARYTM, introduced in 1995. The most recent is BOOLE, implemented in 2001.

[4] Recently, the Library Committee decided to provide a special article covering the proofs of requirements which can be formulated as Mizar statements. The first article of that series is BOOLE, [5].

3.3 requirements SUBSET

This **requirements** directive concerns the definition of inclusion (TARSKI:def 3, [11]), the power set (ZFMISC_1:def 1, [3]) and also mode '**Element of**' (SUBSET_1:def 2) with a following redefinition, [12]). With this directive **X c= Y** automatically yields **X is Subset of Y** and vice versa. The property of the form **x in X & X c= Y implies x in Y** is incorporated as well[5]. When BOOLE is also applied in the **requirements** directive, the formula **x in X** is equivalent to **x is Element of X & X is non empty**.

3.4 requirements ARYTM

Specification of **requirements ARYTM** concerns the definitions provided in the article ARYTM ([7]): the set **REAL** (ARYTM:def 1), the redefinition of the set **NAT** as a **Subset of REAL**, the real addition and multiplication operations (ARYTM:def 3, def 4) and the natural ordering of real numbers (ARYTM:def 5). It provides the correspondence between numerals and numbers defined in the MML. Without **requirements ARYTM** and **SUBSET**, numerals are just names for (not fixed) sets. With them, numerals obtain internally the type **Element of NAT** and appropriate values stored as rational numbers. These values are also used to assign a proper order between numerals. It also makes basic addition and multiplication operations on rational numbers accepted by the checker with no additional justification. The numerators and denominators of Mizar numerals must not be greater than 32767 (the maximum value for a 16-bit signed integer), although all internal calculations are based on 32-bit integers. For example, the following equalities can be easily calculated and therefore they are obvious: **x + 0 = x, x * 0 = 0, x * 1 = x**. More processing capabilities for other arithmetic operations are introduced by the **requirements REAL** directive (see Section 3.5).

3.5 requirements REAL

This enables special processing of real expressions based on the constructors REAL_1:def 1 - REAL_1:def 4, [8]. As with **ARYTM**, the Mizar checker uses the rational value associated with real variables and a built-in GCD routine to evaluate new equalities. In particular, the following equalities are directly calculated at this stage: **x / 1 = x, x - 0 = x**, etc.

4 Conclusions

The above considerations show that there are quite efficient mechanisms in Mizar that provide some elements of computer algebra. The idea of implementing such features was based on statistical observations showing the extensive use of special constructs. The introduction of these techniques has a strong influence on the maintenance of the MML. However, the distinction between the function of properties and requirements is not always clear. In some cases, it is hard to decide which of these techniques is the best implementation. Requirements are much more flexible, but on the other hand, properties are a regular language construct and are not so system dependent. Every Mizar user can decide whether or not to use properties for newly created definitions while a new **requirements** directive yields a partial reimplementation of the system. Some of the features currently implemented as requirements could be transformed into some kind of

[5] Formerly, it was an extensively used MML theorem BOOLE:11.

properties. In particular, this would concern the properties of neutral elements as described in Sections 3.4 and 3.5. There is still discussion on what would be the best syntax for such a **neutrality** property. Another area of interest is the implementation of the **associativity** and **transitivity** properties. However, this work still remains in the to-do list due to some problems with finding an approach that will generate an efficient implementation.

Acknowledgment: The authors would like to express their gratitude to Pauline N. Kawamoto for her kind help in preparation of this paper.

References

[1] Grzegorz Bancerek, *The fundamental properties of natural numbers*, Journal of Formalized Mathematics, http://mizar.org/JFM/Vol1/nat_1.html.
[2] Grzegorz Bancerek, *The ordinal numbers*, Journal of Formalized Mathematics, http://mizar.org/JFM/Vol1/ordinal1.html.
[3] Czesław Bylinski, *Some basic properties of sets*, Journal of Formalized Mathematics, http://mizar.org/JFM/Vol1/zfmisc_1.html.
[4] Library Committee, *Boolean Properties of Sets – Definitions*, Journal of Formalized Mathematics, Encyclopedia of Mathematics in Mizar, 2002, http://mizar.org/JFM/EMM/xboole_0.html.
[5] Library Committee, *Boolean Properties of Sets – Requirements*, Journal of Formalized Mathematics, Encyclopedia of Mathematics in Mizar, 2002, http://mizar.org/JFM/EMM/boole.html.
[6] Library Committee, *Mizar built-in notions*, Journal of Formalized Mathematics, Axiomatics, 1989, http://mizar.org/JFM/Axiomatics/hidden.html.
[7] Library Committee, *Preliminaries to arithmetic*, Journal of Formalized Mathematics, Addenda, 1995, http://mizar.org/JFM/Addenda/arytm.html.
[8] Krzysztof Hryniewiecki, *Basic properties of real numbers*, Journal of Formalized Mathematics, http://mizar.org/JFM/Vol1/real_1.html.
[9] Jan Popiolek, *Some properties of functions modul and signum*, Journal of Formalized Mathematics, http://mizar. org/JFM/Vol1/absvalue.html.
[10] Piotr Rudnicki and Andrzej Trybulec, *On Equivalents of Well-Foundedness. An Experiment in MIZAR,* Journal of Automated Reasoning, 23, 1999, pp. 197-234.
[11] Andrzej Trybulec, *Tarski Grothendieck set theory*, Journal of Formalized Mathematics, http://mizar.org/JFM/Axiomatics/tarski.html.
[12] Zinaida Trybulec, *Properties of subsets*, Journal of Formalized Mathematics, http://mizar.org/JFM/Vol1/subset_1.html.
[13] Freek Wiedijk, *Checker*, available on WWW: http://www.cs.kun.nl/~freek/notes/by.ps.gz.
[14] Edmund Woronowicz, *Relations and their basic properties*, Journal of Formalized Mathematics, http://mizar.org/JFM/Vol1/relat_1.html.