Extending Numeric Automation for Number Theory Formalizations in MIZAR

Adam Naumowicz

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

CICM 2023, September 5, 2023

Outline

1. Introducing an experimental version of the Mizar proof checker equipped with new built-in routines for automating common numeric computations.
2. Evaluating the potential of automating parts of simple number theory proofs in the Mizar Mathematical Library (MML).

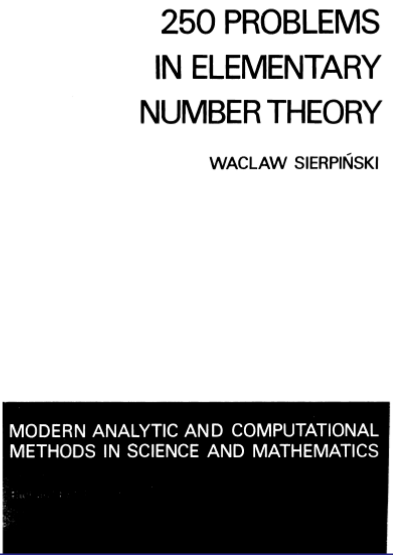
Introduction

- Mizar is a proof assistant best known for:
  - its underlying proof language primarily designed to closely resemble the mathematical vernacular,
  - the pioneering long-term development of a repository of formalized mathematics, Mizar Mathematical Library (MML), established in 1989.
- For several decades the development of MML has been conducted in parallel to the evolution of the proof checking system.
- Since its inception in 1973, Mizar has been designed to be used by mathematicians, not programmers (implementing extensions by typical users is not possible).
- Various features of the system have been implemented by the system developers in response to specific needs that emerged during the formalization of particular theories.

Formalizing Sierpinski's Book

- A formalization project started as a tribute to Wacław Sierpiński aims at encoding his book '250 Problems in Elementary Number Theory'.
- The initial Mizar formalizations related to that effort were done in 2020 and the resulting dataset based on 10 initial problems was presented at CICM 2020.
  Naumowicz, A.: Dataset Description: Formalization of Elementary Number Theory in Mizar. In: Benzmüller, C., Miller, B.R. (eds.): CICM 2020. LNCS, vol. 12236, pp. 303–308. Springer, Heidelberg (2020).
- To date, the joint formalization work resulted in developing a sequence of Mizar articles submitted to MML which cover more than 100 of the problems.

Formalizing Sierpinski's Book

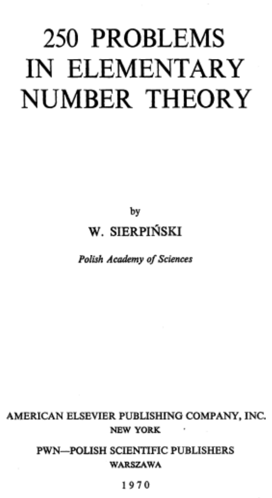


250 PROBLEMS IN ELEMENTARY NUMBER THEORY

WACLAW SIERPIŃSKI

MODERN ANALYTIC AND COMPUTATIONAL METHODS IN SCIENCE AND MATHEMATICS

Formalizing Sierpinski's Book


250 PROBLEMS IN ELEMENTARY NUMBER THEORY

by

W. SIERPIŃSKI

Polish Academy of Sciences

AMERICAN ELSEVIER PUBLISHING COMPANY, INC.
NEW YORK

PWN—POLISH SCIENTIFIC PUBLISHERS
WARSZAWA

1970

Formalizing Sierpinski's Book

- Sierpinski had numerous contributions to many fields of mathematics, but number theory was his first main area of interest.
- Sierpinski's prolific and diverse research resulted in over 700 papers and 50 books.
- The content of the book covers the following chapters:
    - I. Divisibility of Numbers,
    - II. Relatively Prime Numbers,
    - III. Arithmetic Progressions,
    - IV. Prime and Composite Numbers,
    - V. Diophantine Equations,
    - VI. Miscellanea.

Standard Numeric Automation in Mizar

- Mizar checker supports direct calculations on rational complex numbers (complex numbers with rational coefficients).
- This functionality is switched on by inputting the, so called, requirements ARITHM directive in the environ part of the Mizar text being developed.
- Mizar checker performs polynomial elimination and accepts numeric operations as obvious, so the users do not need to provide any justification for the parts of their proofs based on such computations.

NUMBER* and XPRIME* Articles

- Formalizing numerous facts related to elementary number theory, the authors often needed to refer to, e.g., the basic divisibility properties of concrete (sometimes quite big) numbers, or to prove whether a particular number is prime or not.
- To facilitate writing such proofs on top of the current MML, A. Kornilowicz of the Mizar library committee generated a set of 'encyclopedic' articles identifying all prime numbers in the range up to 10,000.
- These articles contain a handy set of referential facts that authors may potentially need when formalizing various theorems in number theory.
- Drawback: the massive files (almost 800K lines of Mizar text in total) cause serious performance problems, especially when processing the whole library is required.
- Solution: our extended automation is devised to eliminate the users' need to directly reference facts from these articles by making them obvious for the Mizar checker.

New Directive: requirements INT_D

- All the presented automatized notions have their definitions in two Mizar articles, (INT_1 and INT_2), so the corresponding new library file dubbed int_d.dre ('d' for divisibility) provides the following signature and links between respective MML constructors and the numbers of the built-in requirement type in the Mizar code:

```
<?xml version="1.0"?>
<Requirements>
<Signature>
<ArticleID name="HIDDEN"/>
<ArticleID name="INT_1"/>
<ArticleID name="INT_2"/>
</Signature>
<Requirement constrkind="K" constrnr="4" nr="35"/>
<Requirement constrkind="K" constrnr="5" nr="36"/>
<Requirement constrkind="R" constrnr="3" nr="37"/>
<Requirement constrkind="K" constrnr="7" nr="38"/>
<Requirement constrkind="K" constrnr="8" nr="39"/>
<Requirement constrkind="V" constrnr="3" nr="40"/>
</Requirements>
```

- The values of the constrnr XML attributes represent the numbering derived from the imported MML signature, whereas the nr attributes refer to hard-coded requirements.

Functors div and mod

- The numeric constant calculations make use of simple routines that compute the div and mod operations that must exactly match the semantics of the library definitions (including the floor operation for div and the mod 0 variant):

```
definition
  let i1,i2 be Integer;
  func i1 div i2 -> Integer equals :: INT_1:def 9
  [\ i1 / i2 /];
  func i1 mod i2 -> Integer equals :: INT_1:def 10
  i1 - (i1 div i2) * i2 if i2 <> 0
  otherwise 0;
end;
```

Functors `lcm` and `gcd`

- Operations for calculating the least common multiple and the greatest common divisor of two integer values must match the general Mizar definitions:

```
definition
  let a,b be Integer;
  func a lcm b -> Nat means :: INT_2:def 1
  a divides it & b divides it &
  for m being Integer st a divides m & b divides m holds
       it divides m;

  func a gcd b -> Nat means :: INT_2:def 2
  it divides a & it divides b &
  for m being Integer st m divides a & m divides b holds
       m divides it;
end;
```

Predicate divides

- The next automatized definition denotes the integer divisibility:

```
definition
  let i1,i2 be Integer;
  pred i1 divides i2 means :: INT_1:def 3
  ex i3 st i2 = i1 * i3;
end;
```

- Such predicative definitions can be automatized using definitional expansions, but then a typical proof context looks this way:

```
    30 = 2*15;
    then 2 divides 30;
```

- Note the lack of references in both proof steps, yet the first inference is necessary to provide the witness for the expansion of the `divides` definition.
- Our automation eliminates the need to input such intermediate steps whatsoever.

Attribute `prime`

- The notion of primality is defined the standard way, but technically applicable to any integer number:

```
definition
  let p be Integer;
  attr p is prime means :: INT_2:def 4
  p > 1 & for n being Nat st n divides p
  holds n = 1 or n = p;
end;
```

- Our automation saves users from having to refer to the encyclopedic articles of the `XPRIME*` collection.
- Note: it might still be worthwhile that the available proofs be maintained by the MML committee as a sort of low-level complete proof data or for regression testing purposes.

Example Problem Statement

- **76.** *Find three least positive integers n such that there are no primes between n and $n + 10$, and three least positive integers m such that there are no primes between $10m$ and $10(m + 1)$.*

-
  ```
  definition
    let n be Nat;
    pred n satisfies_Sierpinski_problem_76a means
    for x being Nat st n < x < n+10 holds x is non prime;
  end;
    let m be Nat;
    pred m satisfies_Sierpinski_problem_76b means
    for x being Nat st 10*m < x < 10*(m+1) holds x is non prime;
  end;

  theorem
    113 satisfies_Sierpinski_problem_76a
    proof
      let x be Nat;
      assume 113 < x < 113+10;
      then 113 < x < 122+1;
      then 113+1 <= x <= 122 by NAT_1:13;
      then x = 114 or ... or x = 122;
      hence thesis by XPRIMES0:114,115,116,117,118,119,120,121,122;
    end;
  ```

Performance Gain

```
[a.naumowicz@esperanto test]$ time ./verifier text/xprimes2.miz
Verifier based on More Strict Mizar Processor, Mizar Ver. 8.1.12 (Linux/FPC)
Copyright (c) 1990-2023 Association of Mizar Users
Processing: text/xprimes2.miz

Parser   [453831]   1:05
MSM      [453830]   1:50
Analyzer [453831]  13:47
Checker  [453831]  1.18:58
Time of mizaring:1.35:40

real    95m41.361s
user    92m24.645s
sys     2m23.118s
[a.naumowicz@esperanto test]$
```

Notes and Conclusions

- The proposed extension of the Mizar system and library can be used by a simple user import command in any Mizar text that develops a theory requiring extensive use of integer divisibility.
- The usefulness of its application is clear from the big number of automated proof steps in typical article-sized Mizar formalizations similar to the `NUMBER*` series.
- Standard Mizar utilities (e.g., `relprem`) equipped with the enhanced checker reveal hundreds of unnecessary references in the original scripts.
- Just as with the other `requirements`, its use should not to imposed on the users, especially if the possibility of developing proofs in full detail may be beneficial, e.g., for didactic purposes.