Teaching How to Write a Proof

Adam Naumowicz

University of Bialystok, Poland

adamn@mizar.org

How can we encode proofs (on computers)?

- Leslie Lamport's "How to Write a Proof"
  - a method of writing proofs which "makes it much harder to prove things that are not true"
  - nowadays numerous computer systems make it "almost completely impossible" to prove things that are not true
- some implications of Freek Wiedijk's comparison of "The Seventeen Provers of the World"
  - no monopoly for the best proof style
  - systems' foundation and philosophy may differ
  - intended goal of creating the proof is important

MIZAR

- proof-checking system (continuously improved since 1970s) aimed at developing formal mathematics in a rigorous way under the control of a computer, but without unnecessary departure from the standard mathematical practice
- underlying proof language close to “mathematical vernacular” based on a declarative style of natural deduction
- long history of using in mathematics instruction (from secondary school courses to writing PhD theses)

MIZAR's distinctive features

- the system is the most "mathematical in spirit"
- readable proof input files
- ZFC set theory and classical logic forming the base
- the use of (dependent-)types
- low but quite efficient automation
- large mathematical standard library (Mizar Mathematical Library - MML)
  - renowned for the biggest library of formalized mathematical data
  - the system has been involved in several big formalization projects
  - attracted almost two hundred authors from more than a dozen countries

Whom to teach writing proofs?

- it has been tried for three decades now to get working mathematicians involved in the use and development of proof-assistants
- Lamport: *Mathematicians tend to be conservative, and many are unwilling to consider that there might be a better way of writing proofs.*
- now we know that it is certainly possible to write fully formal expositions/formalizations of quite advanced mathematics
  - classical theories with well-established theorems
  - specialized monographs
  - recent mathematical journal papers
- the De Bruijn factor still to high to persuade working mathematicians to make this extra effort

If not mathematicians, then who?

- instead of trying to convince this community, there is a new approach worth trying
  - appeal to the new generation
  - today's students don't carry any bias of traditional mathematical practice
  - Lamport's vision:
    - not only pure mathematics matters
    - growing interest in correctness of algorithms
  - computer scientists rather than mathematicians

How difficult is it to learn writing proofs with MIZAR?

- Wiedijk's observation: "proof assistants tend to resemble their implementation language"
- MIZAR is about as complex as Pascal - not complicated at all
- initially Pascal was intended to teach students structured programming
- today Pascal is far too often considered suitable ***only*** for teaching
- NB: although MIZAR possesses certain didactic qualities - it's not only good for teaching!

MIZAR/Pascal similarities

- Object Pascal - 29 special symbols, 65 reserved words and 39 directives
- MIZAR - 27 special symbols, 110 reserved words
  - the numbers almost identical!
  - 10 symbols shared by both languages
  - 15 identical reserved words
  - MIZAR is case-sensitive unlike Pascal

More MIZAR/Pascal similarities

- as many as possible words from natural English
  - both languages are rather high-level
  - construct expressions close to natural language
  - highly structured - to ensure producing rigorous and semantically unambiguous texts
- the way MIZAR reports errors resembles Pascal's errors and warnings
  - "lazy interaction" - step-wise refinement
  - it's possible to check correctness of incomplete texts
  - one can postpone a proof or its more complicated part

MIZAR-aided courses at University of Bialystok

- initially, mathematics department (since 1970s)
- mainly voluntary monographic courses: “Lattice theory”, “Category theory”, “Topology”, etc.
- new CS department emerged - new curriculum created
- undergraduate level courses:
  - “Introduction to formal methods”
  - “Constructive methods in CS”
  - “Abstract methods in CS”
- graduate level courses:
  - “Software verification”
  - “Proof verification”

“Introduction to formal methods”

- logical formulae and basic structures of conditional proofs
- Boolean properties of sets
- families of sets and their properties
- binary relations (composition, the inverse relation, selected properties - e.g. reflexivity, transitivity, etc.)
- functions (domain and codomain, image, etc.)
- equivalence relations, partitions and ordering relations

Example 1: For any relation R, if R is transitive, then $R \circ R \subseteq R$.

```
for R being Relation holds R is transitive implies R*R c= R
proof
  let R be Relation;
  assume a: R is transitive;
  let a,b be set;
  assume [a,b] in R*R;
  then consider c being set such that
  c: [a,c] in R & [c,b] in R by RELATION:def 7;
  thus [a,b] in R by c,a,RELATION:def 12;
end;
```

Example 2: There exist relations R, S and T such that $R \circ (S \setminus T) \not\subseteq (R \circ S) \setminus (R \circ T)$.

```
ex R,S,T being Relation st not R*(S \ T) c= (R*S) \ (R*T)
proof
  reconsider R={[1,2],[1,3]} as Relation by RELATION:2;
  reconsider S={[2,1]}, T={[3,1]} as Relation by RELATION:1;
  take R,S,T;
  b: [1,2] in R by ENUMSET:def 4;
  d: [2,1] in S by ENUMSET:def 3;
  [2,1] <> [3,1] by ENUMSET:2;
  then not [2,1] in T by ENUMSET:def 3;
  then [2,1] in S \ T by d,RELATION:def 6;
  then a: [1,1] in R*(S \ T) by b,RELATION:def 7;
  e: [1,3] in R by ENUMSET:def 4;
  [3,1] in T by ENUMSET:def 3;
  then [1,1] in R*T by e,RELATION:def 7;
  then not [1,1] in (R*S) \ (R*T) by RELATION:def 6;
  hence not R*(S \ T) c= (R*S) \ (R*T) by RELATION:def 9,a;
end;
```

“Constructive methods in CS”

- Peano arithmetic
- various forms of the induction principle
- higher-order reasoning with MIZAR schemes
- the axiomatics of set theory

Example 3: By induction, for natural numbers i, j, k if $i + k = j + k$, then $i = j$.

```
reserve i,j,k,l for natural number;
i+k = j+k implies i=j;
proof
  defpred P[natural number] means i+$1 = j+$1 implies i=j;
  A1: P[0]
  proof
   assume B0: i+0 = j+0;
   B1: i+0 = i by INDUCT:3;
   B2: j+0 = j by INDUCT:3;
   hence thesis by B0,B1,B2;
  end;
  A2: for k st P[k] holds P[succ k]
  proof
    let l such that C1: P[l];
    assume C2: i+succ l=j+succ l;
    then C3: succ(i+l) = j+succ l by C2,INDUCT:4
    .= succ(j+l) by INDUCT:4;
    hence thesis by C1,INDUCT:2;
  end;
  for k holds P[k] from INDUCT:sch 1(A1,A2);
  hence thesis;
end;
```

“Abstract methods in CS”

- Lattice theory
- universal and many-sorted algebra
- elements of category theory

“Software verification”

- various semantics of software description (operational, denotational, axiomatic)
- program correctness criteria
- mathematical models of computers
- practical verification of exemplary algorithms
- generating proof conditions

“Proof verification”

- a bit of formal theory of mathematical proofs
- managing databases of formalized proofs
- practical usage of discussed MIZAR mechanisms
- the objective: to enable carrying out formalization in a specific domain
- the formalization may form a basis of one’s MSc thesis
- students are supposed to be trained enough to produce new contributions to MML

Methodology

- gradual introduction of MIZAR constructs
- proof sketches first
- “active” and “passive” language acquisition (e.g. definitions)
- postponing the use of more high-level features - to enable reflection later on
  - “syntactic sugar” expressions (`then`, `hence`, `thesis`)
  - automatic definition expansion
  - implicit general quantifiers
  - the use of semantic correlates for thesis elimination
  - forward/backward proof distinction
- dedicated (incremented) environments for undergraduate courses
- interacting with the full system for graduate courses

Conclusions

- implementing obligatory curriculum based on MIZAR proved feasible
- the choice of CS students seems prospective (formal methods in software verification and specification)
- the courses are interdependent - learning MIZAR is not a wasted effort!
- teaching process is not harder than that of a popular programming language
- the assessment in a few years' time by the number of MSc theses contributed to MML