Logic as a Path to Enlightenment (Work in Progress Report)

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
Wolfgang.Schreiner@risc.jku.at

Extended Abstract

A core principle of enlightenment is to reject claims that are based on dogma respectively on an appeal to authority (the medieval *ipse dixit* argument: "he (Aristotle) said so"). On the contrary, enlightenment accepts as sources of truth only empirical evidence (observation) and well-formed arguments (reasoning), both of which are freely accessible to every human being. This principle is the source of modernity (and the main difference to the antique/medieval world) and must therefore be in the focus of education of every new generation to be enlightened. This is also of particular importance, because the principle (although it should be self-evident) must be perpetually defended against attacks from many parties (who explicitly or implicitly represent pre- or even anti-modern views). However, it is actually frequently neglected in education, even in "purely rational" disciplines as mathematics or computer science, where (more often than not) students are confronted with propositions, rules, methods, and algorithms whose validity is only justified by ipse dixit: "he (the teacher) said so".

Logic is the science of reasoning and should be therefore considered as a "path to enlightenment", i.e., as a necessary basis for a rational discourse. In particular, the language of predicate logic (introduced in the 19th century by Frege in his "Begriffsschrift") is rich enough to give a precise meaning, not only to most of mathematics, but also to a large part of natural language. This makes it possible to describe complex realities in a precise way as formal models, state propositions in such models as formal sentences, and not only derive a valid argument proving the truth of a proposition, but also judge whether such an argument is indeed valid or not. Teaching the effective use of the language of logic *as a practical working language* for modeling and reasoning about realities is therefore a valuable endeavor. Unfortunately, not many curricula recognize the importance of this goal, partially also because logic is too often presented in a classical "paper and pencil" style as a dry and purely theoretical topic; consequently, this discipline (if being represented at all in a curriculum) is often confined to some "esoteric corner" with little or no relationship to other subjects.

LOGTECHEDU

However, logic has a "trump in the backhand": by essential advances in computational logic (automated reasoning, model checking, satisfiability solving), a considerable part of it can be nowadays well supported (partially completely automated) by computer software. The (well considered and carefully prepared) application of such software may help to demonstrate the practical usefulness of logic and thus increase the motivation of students to study this subject and its relationships to other ones. Most importantly, it allows students to actively engage with the material by solving concrete problems; ultimately, if the software provides adequate feedback, students may be enabled to train themselves in the use of logic as a working tool.

The seed project "LOGTECHEDU: Logic Technology for Computer Science Education" of the Linz Institute of Technology (LIT) at the Johannes Kepler University Linz [LOG18] aims to foster this development in the context of computer science education (with an initial focus on academic undergraduate education that may

In: Proceedings of the Workshop *CME-EI: Computer Mathematics in Education — Enlightenment or Incantation?*, Hagenberg, Austria, 17-Aug-2018, published at http://ceur-ws.org

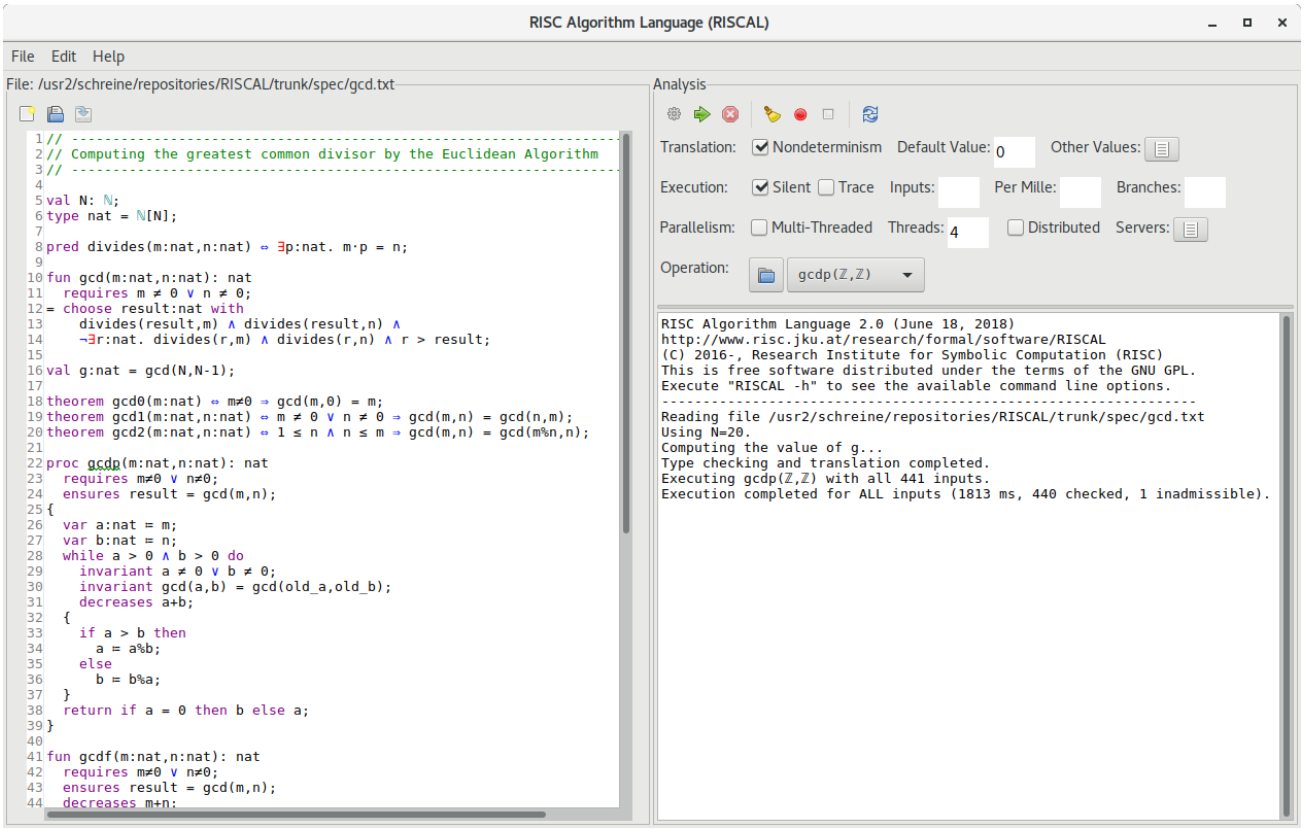

Figure 1: The RISCAL User Interface

later move "upwards and downwards" to graduate education and to high school education). For this purpose, the project simultaneously pursues various research strands on

- Solver Guided Exercises,
- Teaching Solver Technology,
- Proof Assistants for Education (Theorema),
- Specification and Verification Systems for Education (RISCAL), and
- Logic across the Subjects in Primary, Secondary and Higher Education.

The ultimate goal of these strands is to pave the way to a form of *self-directed learning* where the role of the teacher becomes that of an "enabler" that provides the necessary background knowledge and the basic skills to allow students to "educate themselves" by the use of logic-based software for actively solving problems in the form of (voluntary) quizzes, (mandatory) assignments, and possibly even (graded) exams. One of these strands pursued by the author of this abstract is presented below in somewhat more detail.

RISCAL

The *RISC Algorithm Language (RISCAL)* [RIS17, Sch17] is a formal modeling language and associated software system (see Figure 1 for its graphical user interface) that allows the formulation of mathematical models, theorems that are supposed to hold in these models, and algorithms that are expected to solve particular problems in these models, all of this in such a way that errors in models, theorems, and algorithms can be quickly detected in a fully automatic way, *before* correctness proofs are attempted [Sch18]. This work is triggered by the observation that most of the time in proofs (e.g., the proof of the correctness of an algorithm) is wasted in vain because due to errors (in the algorithm's specification, implementation, or meta-knowledge such as loop invariants) the supposed theorem actually does *not* hold. However, from a failed proof attempt (usually performed with the

```
val N: ℕ; type nat = ℕ[N];

pred divides(m:nat,n:nat) ⇔ ∃p:nat. m·p = n;
fun gcd(m:nat,n:nat): nat
  requires m ≠ 0 ∨ n ≠ 0;
= choose result:nat with
    divides(result,m) ∧ divides(result,n) ∧
    ¬∃r:nat. divides(r,m) ∧ divides(r,n) ∧ r > result;

theorem gcd0(m:nat) ⇔ m≠0 ⇒ gcd(m,0) = m;
theorem gcd1(m:nat,n:nat) ⇔ m ≠ 0 ∨ n ≠ 0 ⇒ gcd(m,n) = gcd(n,m);
theorem gcd2(m:nat,n:nat) ⇔ 1 ≤ n ∧ n ≤ m ⇒ gcd(m,n) = gcd(m%n,n);

proc gcdp(m:nat,n:nat): nat
  requires m≠0 ∨ n≠0;
  ensures result = gcd(m,n);
{
  var a:nat := m; var b:nat := n;
  while a > 0 ∧ b > 0 do
    invariant gcd(a,b) = gcd(old_a,old_b);
    decreases a+b;
  {
    if a > b then a := a%b; else b := b%a;
  }
  return if a = 0 then b else a;
}
```

Figure 2: The Greatest Common Divisor and the Euclidean Algorithm

help of an interactive proof assistant) it is hard to see whether the failure is the result of an inadequate proof strategy or of some error in the formal model; this substantially hampers the progress of learning the use of logic for formal modeling and reasoning.

To overcome this problem, RISCAL is based on a restricted form of predicate logic that bounds the size of models by (user-declared) parameters. For every set of concrete values assigned to these parameters, the resulting model instance is finite; this allows to evaluate in that instance all functions and predicates, fully automatically decide the truth of all propositions, and to check the correctness of all algorithms for all possible inputs.

As an example (included in the distribution as file `gcd.txt`, see Section 2 of [Sch17] how to run it), Figure 2 displays a RISCAL model that, in the domain `nat` of all natural numbers less than equal the model parameter N, formulates the theory of greatest common divisors (a predicate `divides`, an implicitly defined function `gcd`, and three theorems `gcd0`, `gcd1`, and `gcd2`); it also defines a procedure `gcdp` that implements Euclid's algorithm to compute the greatest common divisor.

We can quickly check for $N = 20$ that e.g. theorem `gcd2` is valid

```
Executing gcd2(ℤ,ℤ) with all 441 inputs.
Execution completed for ALL inputs (256 ms, 441 checked, 0 inadmissible).
```

and that the procedure `gcdp` satisfies its specification:

```
Executing gcdp(ℤ,ℤ) with all 441 inputs.
Execution completed for ALL inputs (933 ms, 440 checked, 1 inadmissible).
```

If the theorem were not true or the algorithm would violate its specification or loop invariants, these errors would be automatically reported.

Experience shows that errors in theorems and algorithms can be usually quickly detected in small model instances; if such a falsification is not possible, theorems and algorithms are validated, i.e., our confidence in their correctness is so much increased, that we have sufficient incentive to subsequently attempt their proof-based verification for models of arbitrary size (with the help of a suitable proving assistant).

To aid the subsequent proof-based verification of the general correctness of algorithms in models of arbitrary size, RISCAL also includes a verification condition generator that produces logical formulas whose validity implies

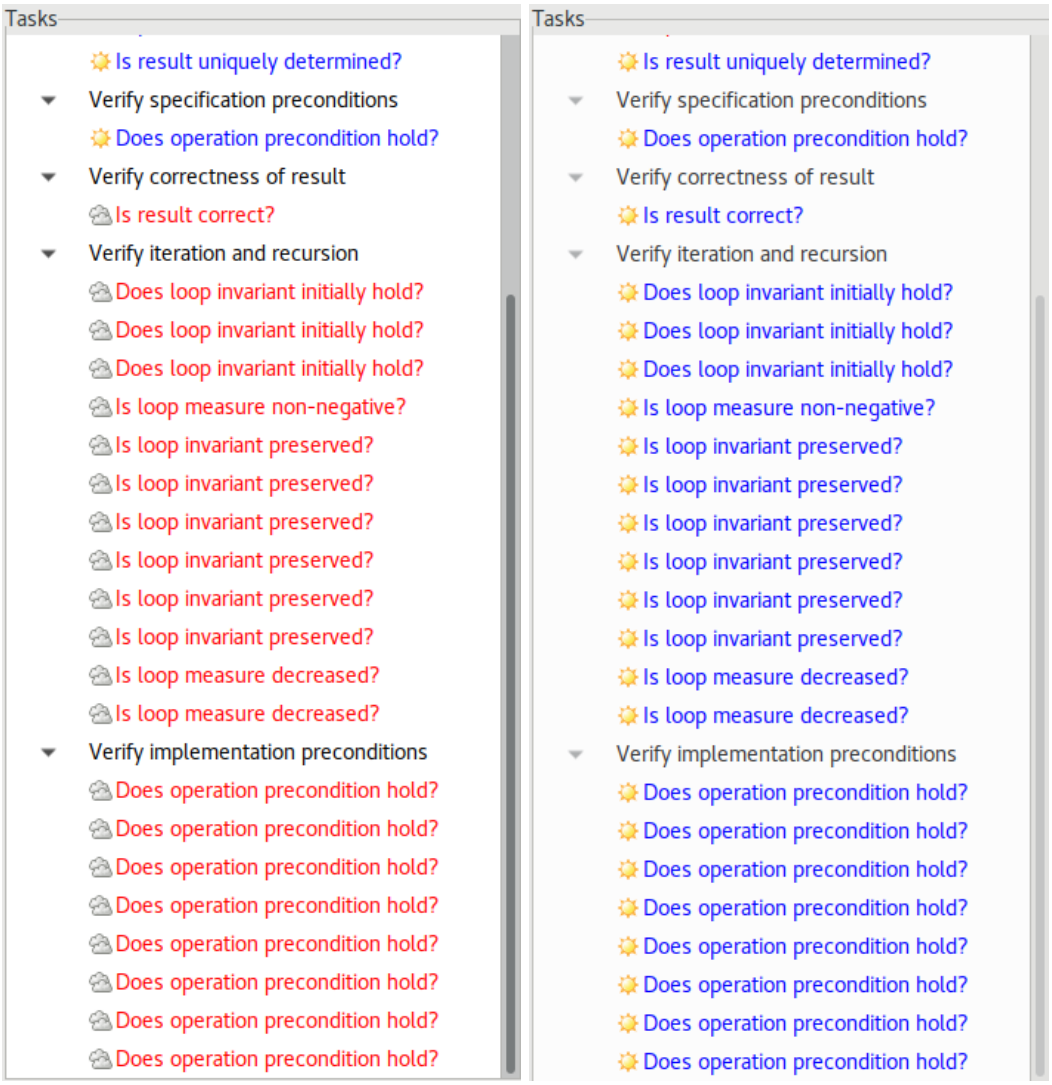

Figure 3: Verification Conditions Before and After their Checking

the correctness of the algorithm; since these formulas are interpreted in the same finite model as the algorithm, they can be checked within the RISCAL software (see Figure 3). If they pass the check, this gives evidence to believe that they the algorithm annotations (loop invariants, etc.) are also sufficient to carry a correctness proof for models of arbitrary size.

In the context of the LOGTECHEDU project, the goal of RISCAL is to develop a systematic catalog of material where students can, in a step-by-step fashion, train their proficiency in the use of formal logic to model realities, state propositions, and specify algorithms; we will elaborate a suitable collection of assignments where the capabilities of the RISCAL software are utilized to give students suitable feedback in the case of errors. This material shall cover selected areas of computer science, (discrete) mathematics, computer algebra, and logic; example content has been (and currently is being) produced in the frame of some bachelor theses [SBF18].

In the mid-term, we plan to utilize this material in (own) courses such as "Logic", "Formal Modeling", and "Formal Methods in Computer Science" reshaping the way how these courses are taught by shifting them towards computer-supported self-directed learning. On the long term, we also hope to convince other lectures in areas such as introductory courses on algorithms and software development; here the initial focus may be to enable students to self-check the correctness of algorithms/programs with respect to given specifications; later the focus may shift towards the formulation of own specifications. We hope to contribute in that way towards an "enlightenment" of students which is based on rational thinking rather than on an uncritical belief in authorities.

Acknowledgments

Supported by the Johannes Kepler University, Linz Institute of Technology (LIT), project LOGTECHEDU, and by the OEAD WTZ project SK 14/2018 SemTech.

References

[LOG18] JKU LIT Project LOGTECHEDU, May 2018. http://fmv.jku.at/logtechedu/.

[RIS17] The RISC Algorithm Language (RISCAL), March 2017. https://www.risc.jku.at/research/formal/software/RISCAL.

[SBF18] Wolfgang Schreiner, Alexander Brunhuemer, and Christoph Fürst. Teaching the Formalization of Mathematical Theories and Algorithms via the Automatic Checking of Finite Models. In *Post-Proceedings ThEdu'17, Theorem proving components for Educational software*, volume 267 of *EPTCS*, pages 120–139, 2018. https://doi.org/10.4204/EPTCS.267.8.

[Sch17] Wolfgang Schreiner. The RISC Algorithm Language (RISCAL) — Tutorial and Reference Manual (Version 1.0). Technical report, RISC, Johannes Kepler University, Linz, Austria, March 2017. Download from [RIS17].

[Sch18] Wolfgang Schreiner. Validating Mathematical Theories and Algorithms with RISCAL. In F. Rabe, W. Farmer, G. Passmore, and A. Youssef, editors, *CICM 2018 — 11th Conference on Intelligent Computer Mathematics, August 13–17, 2018*, volume 11006 of *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence*, Hagenberg, Austria, 2018. Springer, Berlin. https://doi.org/10.1007/978-3-319-96812-4_21.