LOGIC AS A PATH TO ENLIGHTENMENT (WORK IN PROGRESS REPORT)

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria

Enlightenment and Education

- Enlightenment: reject claims based on authority (“ipse [Aristotle] dixit”)
  - Only two sources of truth acceptable:
    - Empirical evidence (observation)
    - Well-formed arguments (reasoning).
  - Stark contrast to pre- or even anti-modern views.
- Education: often claims accepted by authority (“ipse [the teacher] dixit”)
  - Even in “rational” disciplines like mathematics or computer science.
    - Presentations of propositions, rules, methods, and algorithms (more often than not) lack proper justification.
  - Students educated to become “believers” (or, equally worse, “non-believers”) rather than “rational skepticists”.

Students should be provided a basis for rational discourse.

Logic as a Path to Enlightenment

Logic as the “science of reasoning” provides such a basis.

- Predicate logic: the “modern” logic of today.
  - Starting with Frege’s “Begriffsschrift” in 1879.
  - Incorporates and supersedes Aristotle’s term logic.
  - Rich enough to capture most of mathematics and much of natural language.
- Construct formal models of reality with precise meaning and reasoning rules.
  - State propositions as formal sentences.
  - Derive valid arguments that prove the propositions.
  - Judge whether such arguments are valid or not.

Should be taught as a practical “working language” for modeling and reasoning.

Educating with the Help of Logic

Do not consider logic just as a “paper and pencil” topic.

- Today much of logic can be automated by computer software.
  - Advances in computational logic (automated reasoning, model checking, satisfiability solving) may aid in many activities.
- Education may be supported by the application of such software
  - Needs to be well considered and carefully prepared.
  - May demonstrate the practical usefulness of logic.
  - May increase the motivation of students to model and to reason.

Core idea: let students actively engage with lecturing material by solving concrete problems and by receiving feedback from software.

JKU LIT Project “LogTechEdu”

Johannes Kepler University Linz Institute of Technology project
“Logic Technologies for Computer Science Education” (2018–2020):

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

Joint activities of the JKU institutes FMV (Biere, Seidl) and RISC (Schreiner, Windsteiger) and supported by the Linz School of Education (Sabitzer).

The RISC Algorithm Language (RISCAL)

A language and software system for investigating finite mathematical models.

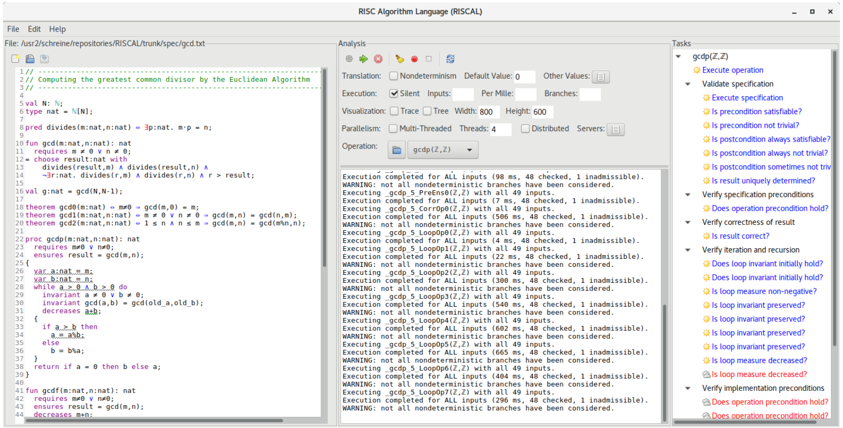
- Formulation of mathematical theories and theorems.
- Formulation and specification of (also non-deterministic) algorithms.
- Rooted in strongly typed first order logic and set theory.
- All types are finite (with sizes determined by model parameters).
- All formulas are automatically decidable by model checking.
- Correctness of all algorithms is automatically decidable by model checking.

Checking in some model of fixed size before proving in models of arbitrary size.

RISCAL Specifications

```
val n: ℕ; val cn: ℕ;
...
type Formula = Set[Clause];
type Valuation  = ClauseBase with clause(value);
...
pred satisfiable(f:Formula) ⇔ ∃v:Valuation. satisfies(v,f);
pred valid(f:Formula) ⇔ ∀v:Valuation. satisfies(v,f);
fun not(f: Formula):Formula = { c | c:Clause with ∀d∈f. ∃l∈d. -l∈c };
...
theorem notValid(f:Formula) ⇔ valid(f) ⇔ ¬satisfiable(not(f));
...
multiple pred DPLL(f:Formula)
  ensures result ⇔ satisfiable(f); decreases |literals(f)|;
⇔ if f = ∅[Clause] then
    ⊤
  else if ∅[Literal] ∈ f then
    ⊥
  else choose l∈literals(f) in
    DPLL(substitute(f,l)) ∨ DPLL(substitute(f,-l));
```

The RISCAL Software



RISC Algorithm Language (RISCAL)

File Edit Help

File: /usr2/schreine/repositories/RISCAL/trunk/spec/gcd.txt

```
// ---------------------------------------------------------------
// Computing the greatest common divisor by the Euclidean Algorithm
// ---------------------------------------------------------------

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;

val g:nat = gcd(N,N-1);

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 a ≠ 0 ∨ b ≠ 0;
    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;
}

fun gcdf(m:nat,n:nat): nat
  requires m≠0 ∨ n≠0;
  ensures result = gcd(m,n);
  decreases m+n;
```

Analysis

Translation: ☐ Nondeterminism Default Value: 0 Other Values:

Execution: ☑ Silent Inputs: Per Mille: Branches:

Visualization: ☐ Trace ☐ Tree Width: 800 Height: 600

Parallelism: ☐ Multi-Threaded Threads: 4 ☐ Distributed Servers:

Operation: gcdp(ℤ,ℤ)

```
Execution completed for ALL inputs (98 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing _gcdp_5_PreEns0(ℤ,ℤ) with all 49 inputs.
Execution completed for ALL inputs (7 ms, 48 checked, 1 inadmissible).
Executing _gcdp_5_CorrOp0(ℤ,ℤ) with all 49 inputs.
Execution completed for ALL inputs (506 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing _gcdp_5_LoopOp0(ℤ,ℤ) with all 49 inputs.
Execution completed for ALL inputs (4 ms, 48 checked, 1 inadmissible).
Executing _gcdp_5_LoopOp1(ℤ,ℤ) with all 49 inputs.
Execution completed for ALL inputs (22 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing _gcdp_5_LoopOp2(ℤ,ℤ) with all 49 inputs.
Execution completed for ALL inputs (300 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing _gcdp_5_LoopOp3(ℤ,ℤ) with all 49 inputs.
Execution completed for ALL inputs (540 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing _gcdp_5_LoopOp4(ℤ,ℤ) with all 49 inputs.
Execution completed for ALL inputs (602 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing _gcdp_5_LoopOp5(ℤ,ℤ) with all 49 inputs.
Execution completed for ALL inputs (665 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing _gcdp_5_LoopOp6(ℤ,ℤ) with all 49 inputs.
Execution completed for ALL inputs (404 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing _gcdp_5_LoopOp7(ℤ,ℤ) with all 49 inputs.
Execution completed for ALL inputs (296 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
```

Tasks

- gcdp(ℤ,ℤ)
  - Execute operation
  - Validate specification
    - Execute specification
    - Is precondition satisfiable?
    - Is precondition not trivial?
    - Is postcondition always satisfiable?
    - Is postcondition always not trivial?
    - Is postcondition sometimes not triv
    - Is result uniquely determined?
  - Verify specification preconditions
    - Does operation precondition hold?
  - Verify correctness of result
    - Is result correct?
  - Verify iteration and recursion
    - Does loop invariant initially hold?
    - Does loop invariant initially hold?
    - Is loop measure non-negative?
    - Is loop invariant preserved?
    - Is loop invariant preserved?
    - Is loop invariant preserved?
    - Is loop invariant preserved?
    - Is loop measure decreased?
    - Is loop measure decreased?
  - Verify implementation preconditions
    - Does operation precondition hold?
    - Does operation precondition hold?

Automatic checking of theorems, algorithms, and verification conditions.

Demonstration

An Exercise in Formal Problem Specification

Conclusions

- Goal: logic-based self-directed learning
  - Techer become “enablers” by providing basic knowledge and skills
  - Students “educate themselves” by solving problems.
    - (Voluntary) quizzes, (mandatory) assignments, possibly (graded) exams.
- Initial target: undergraduate university students.
  - Reachout both “up and down” to graduate students and to high-school students.
- Initial focus: computer science and mathematics.
  - First own courses on “Logic”, “Formal Modeling”, “Formal Methods”; later also others’ introductory courses on algorithms and software development.

Towards “enlightenment” via “rational thinking” by “self-directed learning”.