Deductive Program Verification: Mature Enough to be Taught to Software Engineers?

Marc Schoolderman, Sjaak Smetsers, Marko van Eekelen
18 November 2019
A short exercise in epistemology

What is the correct answer?

\[14627333968688430831 \times 18369543843582177293 = 268697452652541021050625400954609320483 ?
\]
\[= 268697426686324666285319976061674831483 ?
\]
Computers do make mistakes

1994: Intel Pentium: \texttt{FDIV} instruction infamously “flawed”

\[
\frac{4195835}{3145727} = 1333820449136241002 \ldots \text{ or } 1333739068902037589 ?
\]

1995: AMD K5: \texttt{FDIV} instruction verified using formal methods
  - Division microcode complies with IEEE-754 standard
    - \textit{Proof by: J Strother Moore, T. Lynch, M. Kaufmann}
  - ACL2 interactive theorem prover
    - \textit{Developed by: J Strother Moore, M. Kaufmann}
Does this compute $X_{64} \times Y_{64}$?

2015: High-performance multiplication for AVR microcontrollers (M. Hutter, P. Schwabe, 2015)
How difficult is program verification, really?

Experiences from (trying to) do research in cooperation with industry:

• Gap between industry and academia
• Tools used by academics (or are perceived to be) too esoteric:
  – “Show me something an educated software engineer could use.”

For wider adaption, this gap needs to be bridged.
Overview of the Why3 Verification Platform

(J.C. Filliâtre, F. Bobot, C. Marché, G. Melquiond, A. Paskevich)
Overview of the Why3 Verification Platform

(J.C. Filliâtre, F. Bobot, C. Marché, G. Melquiond, A. Paskevich)
Participants in the course “Software Analysis”

22 actively participating students

- Possess a Bachelor degree: university/vocational university (HBO)
  - Similar to many junior software engineers?
- Expected: little/no experience in formal verification
  - Students from Radboud have seen some model checking, and pen-and-paper Hoare logic
  - JML used for 1 short exercise in parallel course

- Goal: evaluate Why3
  - Learning by doing, teamwork, open problems.
Course structure

• **Lectures** (6 hours)
  1) Motivation for verification, introduction of Why3
  2) Why3 data type system
  3) Techniques to work around “stuck” proof efforts
  4) WhyML as a modelling language

• **Small exercises** (20 hours)
  - Supporting the lectures, formative feedback

• **Verification task** (24–36 hours)
  - Report + evaluation of Why3
Case study 1: safe string concatenation (taken from CloudLibc)

```c
size_t strlcat(char *restrict s1, const char *restrict s2, size_t n) {  
  size_t skipped = 0;
  while (n > 0 && *s1 != '\0') {  
    ++s1;
    --n;
    ++skipped;
  }
  const char *begin = s2;
  while (n > 1) {  
    *s1 = *s2;
    if (*s2 == '\0')
      return s2 - begin + skipped;  
    ++s1;
    ++s2;
    --n;
  }
  if (n > 0)
    *s1 = '\0';
  while (*s2 != '\0')
    ++s2;
  return s2 - begin + skipped;
}
```

**Challenges:**

- Arguments must not alias
- “Safety valve” prevents out-of-bounds access, but breaks the naive contract.
- Null-terminated strings
void add(int64_t o[16], int64_t a[16], int64_t b[16])
{
    // add limbs
    for(int i=0; i<16; i++) {
        o[i] = a[i] + b[i];
    }

    // carry propagation
    for(int i=0; i<15; i++) {
        int64_t c = o[i] >> 16;
        o[i+1] = o[i+1] + c;
        o[i] = o[i] - (c << 16);
    }

    // reduce mod 2^255 - 19
    int64_t c = o[15] >> 16;
    int64_t t = 38*c;
    o[0] = o[0] + t;
    o[15] = o[15] - (c<<16);
}

Challenges:
• Weird representation of integers
• Unspecified what this is supposed to do
• The third comment lies
Case study 1 results:

• 7 teams were successful
  − Formal specification of `strlcat`, verified.
• 2 teams ran into difficulties
  − Likely cause: poor initial modelling choices

Case study 2 results:

• 1 team: verified mathematical correctness of `add`
• 1 team: proved that the final loop is seldom necessary
  − both teams quickly proved absence of signed integer overflow,
    and discovered known flaws in this code
Subjective observations

• Most effort required: modelling C code in WhyML
  – *Also due to unfamiliarity with C...*

• Students can deal with concepts unique to Why3
  – E.g. “proof context size”, “ghost code”, “logic functions”

• Student remarks in reports (paraphrased):
  – *Positive:* “Why3 is intuitive and gives strong guarantees.”
  – *Negative:* “This was very time-consuming.”
  – *Verdict:* “Useful for safety-critical software, overkill elsewhere.”
Evaluation of Why3 by students

Common theme: more attention to user-friendliness!

- Error messages should be more helpful
- Better counterexample generation
- Automatic warnings about logical inconsistencies
- Single-step integrated debugger
- “Trivial” loop invariants should be generated automatically
- Online support community (e.g. StackExchange)
- Usable on Windows instead of only Linux and MacOS
Research questions:

- Are these students really representative for software engineers?
- How much time did the students need, quantitatively?
- Do the reports contain honest evaluations?
Anonymous research survey

15 respondents out of 22 active students: 68% response ratio

Limitations:
- No open questions
- Not enough to do statistics on

Primary goal:
- Test assumptions about student population
- Validate the evaluation in the report
Anonymous research survey: results

Student background

- Consider themselves fairly skilled programmers
- Clearly not “pure logicians”
- Only two students report taking more than 30EC of math/logic courses
Anonymous research survey: results

60% of students did not know what ‘Hoare logic’ meant!

Were you familiar with Hoare logic before taking this course?
15 responses

- Yes, and this was important while learning Why3.
- Yes, but it made only a little difference.
- No.
- No, and I still am not familiar.
Anonymous research survey: results

Time spent on project:

Worst case scenario:

- Assume: all 6 students that were not successful are in this dataset
- Then: successful *teams* needed on average $\leq 26$ hours
Anonymous research survey: results

What was it like for students?
- “Somewhat harder” and more mathematical than ordinary programming

Hardest activities:
1) Finding loop invariants
2) Modelling a C program
3) Writing a formal specification
Anonymous research survey: results

What do students think about applying formal methods?

*Formal methods are appropriate for:*
  - Small security-critical libraries, programming language design

*No clear consensus:*
  - Self-driving cars, compilers, operating systems

*Formal methods are not appropriate for:*
  - Smartphone apps

- Consistent with the non-anonymous reports
Conclusion

Novices can apply Why3 usefully in a short amount of time
- Verifying small but real program code
- Four weeks of training, ~26 hours of work
- Background: comparable to junior software engineers

Formal tools can benefit from a fresh perspective
- Problem may be *usability* instead of inherent difficulty

Thank you for your attention!