Proofs, Provers, Processes at Scale

N. Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

Oct. 9, 2019

Ada Lovelace and Turing

The science of operations, as derived from mathematics more especially, is a science of itself, and has its own abstract truth and value.

It is of course important that some efforts be made to verify the correctness of assertions that are made about a routine. There are essentially two types of method available, the theoretical and the experimental. In the extreme form of the theoretical method a watertight mathematical proof is provided for the assertion. In the extreme form of the experimental method, the routine is tried out on the machine with a variety of initial conditions and is pronounced fit if the assertions hold in each case.

Alan Turing (quoted by D. MacKenzie in *Risk and Reason*)

Scale in Perspective

- Formal methods span a spectrum of problems and solutions.
- Scale can involve
  1. Problem size
  2. Cost
  3. Performance
  4. Productivity
  5. Usability
  6. Maintainability
  7. Degree of automation
  8. Degree of specialization
  9. Expressiveness
  10. Abstraction level
  11. Background libraries

Formal Methods in the Spectrum

- Methods like static and dynamic program analysis validate/falsify small properties of big systems through aggressive over/under-approximation.
- Model checking algorithmically verifies/falsifies limited properties of models of systems and is quite effective for bug-finding.
- Automated synthesis techniques can target small programs or small pieces of larger programs.
- Program derivation can handle larger programs but requires manual guidance (which is not a bad thing).
- Program verification can require a significant annotation and proof overhead.

What is Achievable Now?

- The seL4 project estimates that it takes about 3 to 4 weeks per 1000 lines of proof.[1]
- The project produced 10KLOC with about 200KLOP (kilo-lines of proof) at 18 person-years.
- Even though lines of proof is not a meaningful metric, this seems like a good ROM estimate.
- In terms of (equivalent) lines of C code, if one roughly estimates that each line of code requires an average (with a substantial variance) of 10 lines of proof (seL4 was 20), verified code productivity can be around 7 KLOC/year.
- For critical system code, productivity levels are around 4 to 7KLOC.
- However, proof maintenance can be considerably harder than code maintenance.

[1] Productivity for Proof Engineering Mark Staples, Ross Jeffery, June Andronick, Toby Murray, Gerwin Klein, Rafal Kolanski, ESEM'14

Looking Ahead: What Scales?

Abstraction scales

- Engineers are used to working with abstractions in the form of models.
- Formal reasoning must also move up the value chain to capture models.
- These models can be easily supported by code generation to multiple languages and platforms.
- Proofs and counterexamples are easier to construct and maintain since abstract models support proof patterns and effective automation through decision procedures and model checking.
- MATLAB Simulink/Stateflow is an example of the success of model-based software engineering.

Looking Ahead: What Scales?

Abstraction scales

- Engineers are used to working with abstractions in the form of models.
- Formal reasoning must also move up the value chain to capture models.
- These models can be easily supported by code generation to multiple languages and platforms.
- Proofs and counterexamples are easier to construct and maintain since abstract models support proof patterns and effective automation through decision procedures and model checking.
- MATLAB Simulink/Stateflow is an example of the success of model-based software engineering.

Looking Ahead: What Scales?

Abstraction scales

- Engineers are used to working with abstractions in the form of models.
- Formal reasoning must also move up the value chain to capture models.
- These models can be easily supported by code generation to multiple languages and platforms.
- Proofs and counterexamples are easier to construct and maintain since abstract models support proof patterns and effective automation through decision procedures and model checking.
- MATLAB Simulink/Stateflow is an example of the success of model-based software engineering.

Looking Ahead

- For a more scalable success story for model-based approaches, we need systematic ways of mapping highly reusable mathematical models to executable code/hardware in domains such as
  - Grammars and parsers
  - Compilers
  - Cryptography
  - Cyber-physical systems
  - Signal processing
  - Machine learning
  - File systems, and
  - Cryptographic/Distributed protocols.
- Within a decade, domain experts can be expected to create tens of thousands of (equivalent) lines of code a year.
- The bigger challenge is ridding computing of its *original sins* so that formal claims can be supported with *efficient arguments* whose cost can be amortized through reuse.