Validating Semantics

Joe Kiniry
Josu Martinez and Martin Hansen

IT University of Copenhagen
IFIP WG 1.9/2.15
14-15 December 2011

Verification Bigots are for the Birds

or

You Must Write and Execute Programs

Motivation

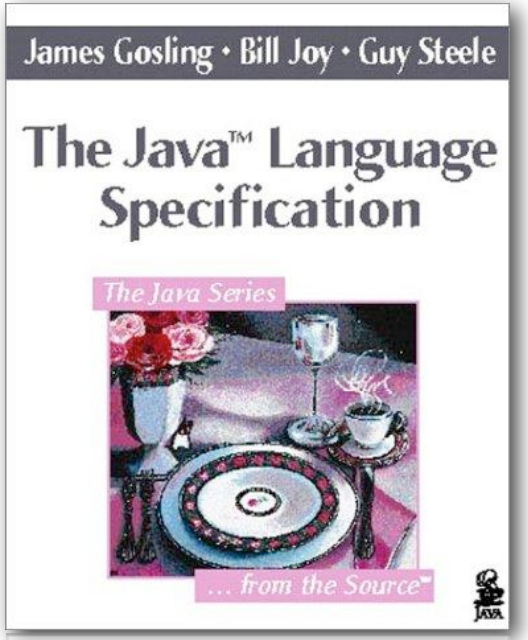
- many “verified” systems contain trivial errors easily caught with “lesser” techniques
- disbelief about many “formal” papers with no mechanization
- disconnect between formality and practice
- missing brutal honesty about soundness and completeness of tools, methodologies, and formalizations
- time/cost/skill limitations of practitioners

Related Work

- Haskell's quickcheck
- PVS random testing
- Microsoft Research's PEX
- ETHZ's AutoTest
- relationship between JML RAC and ESC
- Univ. of Washington, Tacoma's JMLunitNG

Back Story

852 pages!



- LOOP
  - 38K lines of PVS
- Jack
  - ~1,300 Java classes, ...
- ESC/Java2
  - ~1,500 Java classes, ~150 axioms, ~2,500 lines of Simplify, ~2,800 lines of PVS, ~400 lines of SMT-LIB, ~2,000 lines of Coq

Open Questions

- what fragment of the language is meant to be formalized properly?
- where and how is over-approximation used?
- where and how are inappropriate type and term mappings used?
- where and how have simplifications been made with a goal toward usability?
- toward "elegance"?

Secret Sauce

- treat a formal system as a system-under-test
- use traditional techniques from validation to exercise a theory to determine if its meaning matches reality
- automatically identify interesting values and types to check that a theory and its instantiation agree to build evidence that a formalization is correct in the Real World

A Small Example

- BSc course teaching techniques in 2011
- students write formal specifications of traditional ADTs in C# with Code Contracts
  - parameterized Stack, Set, List, and Bag
- use PEX to generate method-level whitebox unit tests (typically obtains ~90% coverage)
- hand-write the minimal number of class-level whitebox unit tests to obtain full "theory coverage" (to obtain >95% coverage)

Large Example

- validating Java semantics
- refinement relation coarsely looks something like "JLS—formalization—VM"
- use the Java grammar to generate small, legal programs with assertions
- check that assertions are valid using formal reasoning infrastructure
- execute programs to check VM behavior

General Methodology

- independently generate a term T and its proposed meaning [[T]]
- use the reasoning infrastructure (the theory-under-test) to check that T agrees with [[T]]
- execute T to obtain a concrete meaning <<T>> from the compiler/runtime/VM/etc.
- compare [[T]] and <<T>> (perhaps using a solver or prover)

FReSH Theory

- concretize hand-waving over autonomous systems development
- apply formal methods techniques to self-* system properties, in particular, self-healing
  - what can one do if a subsystem disappears while a system is executing?
- solution requires a novel combination of orchestration languages, abstract state-based contracts, and specification matching

JMLing Orc

- start with Misra and Cook's Orc language
- describe meaning of Orc sites using abstract state-based contracts
- extend notion of Orc site to include non-pure methods with footprints
- develop composition theory that explains the meaning of the composition of any two Orc programs using Orc's composition operators (sequential, parallel, and pruning composition)

Timeline

- months on paper developing theory
- months mechanizing foundations necessary for mechanizing theory in PVS (e.g., all of Orc and its denotational semantics)
- ...but only completed proofs of around half of the ~125 theorems
- student has finite time and expertise

Validating Semantics

- develop an abstract validation framework for validating contract- and OO-centric theories using Java, JML, and JMLunitNG
- extend this abstract framework into a concrete variant that expresses the FReSH compositional theory
- express the meaning of each rule in the theory via a JML specification
- use JMLunitNG to automatically generate a test framework
- execute the tests to determine if the theory's predictions match the reality of the execution

Other Examples

- algebraic and coalgebraic theories
  - does your Stack implementation conform to the theory of algebraic stacks?
- type systems
  - does your type checker actually conform to your on-paper type system?
- operational and denotational semantics
  - does your semantics match the behavior of the compiler/runtime/VM?