On Formal Definition and Analysis of Formal Verification Processes: A Prospectus

A presentation in honor of Prof. Kokichi Futatsugi Kanazawa, Japan April 14, 2014

Leon J. Osterweil
Lab. For Advanced Software Engineering Research
School of Computer Science
University of Massachusetts
Amherst, MA 01003 USA
The Underlying Problem

• The use of software is now all-pervasive
  – Especially in critical infrastructure
• The quality, reliability, security of this software remains problematic
• Not enough top-quality developers to build it all
• How to increase the quantity of top-quality software and top-quality developers?
  – And increased societal trust, confidence in it
Some Approaches to “Confidence Incrementation”

- Dynamic analysis
  - Primarily Testing
- Static analysis
  - Syntax checking
  - Semantic checks
  - Model checking
- Formal Verification

Combinations of the above aimed at buying increased confidence
At prices that are consistent with the value of the increased confidence
Formal Verification

• The “Gold Standard” for assuring top quality
• Successful formal verifications assure that software “does what it is supposed to do”
  – Although sometimes difficult to know what that is
• More important: FV is a disciplined structure for thinking and reasoning about software
• Leads to
  – Formal proofs of software properties
  – Deeper understanding and confidence in software
  • But for whom?
Drawbacks

• Formal Verification is difficult
• Currently it is largely the domain of experts
• Proofs are hard for many to understand
• Formal Verification process can be tricky
  – How to be sure it has been done right?
  – How to convince various stakeholders?
    • Some of whom are not technically savvy
Some drawbacks have been addressed

- Proof checkers
- Proof scorers
- Proof assistants
- Process assistance tools
  - E.g. Isabelle/HOL
- Formal authorities
  - For proof certification
But some issues remain

• How to assure diverse stakeholders that the Formal Verification can be trusted?
  – In view of the diversity of stakeholder groups
    • Ranging from users and “innocent bystanders” to verifiers themselves
  – Especially in view of evolution, product lines

• How to broaden the community of verifiers?
  – Make good people better
  – Broaden the community
    • Especially to include students who grow into developers of key infrastructure software?
A Modest Proposal--Exploit the fact that: Verification is a Process

• Define it formally
• In a formally defined, executable language
• Formal language definition supports analysis
  – To support credibility of proofs that are generated
• Executable
  – To guide experts
  – To support novices
  – To Broaden, democratize the verification community
But Verification is a Tricky Process

• Nominal process can be described straightforwardly
  – Create and place assertions
  – Prove lemmas
  – Demonstrate termination

• Complexity arises from rework/iteration, due to
  – Incorrect assertions
  – Improper placement of assertions
  – Errors in proofs
  – Inherent difficulty of some proofs
  – Erroneous assumptions during rework

• Existing tools help, but need to be marshalled into an overall verification process strong in supporting the needed iteration/rework
Need a Process Language that can

• Define nominal process clearly, but also:
• Handle non-nominal cases arising from exceptions
  – And long chains of exceptions
• Manage myriad verification process artifacts
• Integrate the contributions of verification assistance tools
• Support reasoning about processes defined in it
• Support execution of these processes
Need a Process Language that can

• Define nominal process clearly, but also:
  • Handle non-nominal cases arising from exceptions
    – And long chains of exceptions
  • Manage myriad verification process artifacts
  • Integrate the contributions of verification assistance tools
  • Support reasoning about processes defined in it
  • Support execution of these processes

Requires special focus on “rework”
What Do I Mean by “Rework”

• Returning to an earlier phase or activity to revisit a decision that turned out to be problematic

• In Formal Verification, could be:
  – Wrong, or problematic, invariant assertion
  – Wrong, or problematic placement of assertion
  – Incorrect or unproveable lemma

• Requires restoring state at the time of the original decision; revisiting decision; continuing
  – Perhaps several times, iteratively
  – Perhaps concurrently
Using Little-JIL to Define a Rework-Intensive Formal Verification Process

• The Little-JIL language is:
  – Formally defined using Finite State Machines
  – Executable
  – Has a graphical user interface
  – Powerful exception management facilities
  – Powerful abstraction capabilities
  – Integrates human and software tools as agents
  – Powerful artifact specification and management

• Has been used to define processes in
  – Healthcare
  – Government
  – Scientific data processing
  – Software development
The Little-JIL Language Iconography
A Little-JIL Definition of Floyd’s Method of Inductive Assertions
Rework Steps—Implemented by Recursion
Rework Steps—Implemented by Recursion
Rework Steps—Implemented by Recursion

Parameters passed include history of all exceptions thrown and how they were handled.
Artifact Evolution History Captured by Data Derivation Graph (DDG)

• Shows which artifacts derived by which process steps, using which process artifacts
• Is a rigorous representation of complete evolution history
• Accessible from the process itself
• Enables human verifier to benefit from previous decisions and their outcomes
Key Questions about FV Process

• Is it verifiable?
• Is it verified?
  – Does it produce correct assurances that all possible verifications are indeed done correctly?
• Is it usable?
  – By experts
  – By novices
• Are its certification results accepted by various stakeholder groups?
• Will it lead to larger quantities of higher quality software and software developers?
Key Questions

• Is it verifiable?
  Yes, using Little-JIL formal semantics

• Is it verified?
  – Does it produce correct assurances that all possible verifications are indeed done correctly?

• Is it usable?
  – By experts
  – By novices

• Are its certification results accepted by various stakeholder groups?

• Will it lead to larger quantities of higher quality software and software developers?
Key Questions

• Is it verifiable?

• Is it verified?
  – Does it produce correct assurances that all possible verifications are indeed done correctly?

• Is it usable?
  – By experts
  – By novices

• Are its certification results accepted by various stakeholder groups?

• Will it lead to larger quantities of higher quality software and software developers?
Key Questions

• Is it verifiable?

• Is it verified?
  – Does it produce correct assurances that all possible verifications are indeed done correctly?

• Is it usable?
  – By experts
  – By novices

• Are its certification results accepted by various stakeholder groups?

• Will it lead to larger quantities of higher quality software and software developers?
Key Questions

• Is it verifiable?

• Is it verified?
  – Does it produce correct assurances that all possible verifications are indeed done correctly?

• Is it usable?
  – By experts
  – By novices

• Are its certification results accepted by various stakeholder groups?

• Will it lead to larger quantities of higher quality software and software developers?

Some Progress: Some asserted properties have been verified
Some Asserted Properties

• All loops broken by an assertion
• All lemmas created
  – And based upon current assertions and code
• All lemmas have been proven
  – And based upon current assertions and code
Some Asserted Properties

• All loops broken by an assertion
• All lemmas created
  – And based upon current assertions and code
• All lemmas have been proven
  – And based upon current assertions and code
Some Asserted Properties

• All loops broken by an assertion
• All lemmas created
  – Based upon current assertions and code
• All lemmas have been proven
  – Based upon current assertions and code

Are complicated by possibilities of many kinds of rework
Approaches to Checking Properties
(Increasing Confidence in the Verification Process)

• Dynamic Analysis
  – Runtime checking of assertions
  – Assisted by careful recording of state and history
    • Using Data Derivation Graph (DDG)

• Finite State Verification/Model Checking
  – Based upon annotated flowgraph of process
  – Automatically generated (using Bandera)
  – Makes critical use of step parameters, instantiated differently for each recursive reference at each rework step.

• Formal Verification
  – Of the Formal Verification Process (!)
Key Questions

- Is it verifiable?
- Is it verified?  
  - Does it produce correct assurances that all possible verifications are indeed done correctly?
- Is it usable?  
  - By experts
  - By novices
- Are its certification results accepted by various stakeholder groups?
- Will it lead to larger quantities of higher quality software and software developers?

Some Progress: Some asserted properties have been verified
But only a small number, at present and NOT Formally Verified
What About the Rest?

• Is it verifiable?
• Is it verified?
  – Does it produce correct assurances that all possible verifications are indeed done correctly?
• Is it usable?
  – By experts
  – By novices
• Are its certification results accepted by various stakeholder groups?
• Will it lead to larger quantities of higher quality software and software developers?
Research Interruptus!

• Graduate Student doing this work left suddenly
  – Gone to Google
  – 😞

• Hoping to pass this on to a new student
• Hoping to build community support for this
• Hoping for fuller report on the occasion of the
  70th year of Prof. Kokichi Futatsugi
Thank you!

Questions?