Software Verification and System Assurance

John Rushby
Based on joint work with Bev Littlewood (City University UK)

Computer Science Laboratory
SRI International
Menlo Park CA USA
Introduction

- **Verification** was the original goal of formal methods
  - Proving correctness of programs
- **Nowadays, overtaken by more pragmatic goals**
  - Generating test cases, synthesizing monitors, bug finding
  - Verifying limited properties: static analysis, extended type checking
- Also, the whole idea of verification has come into question
  - “Proving the absence of errors” is an overly strong claim
  - Lots of caveats about assumptions, fidelity of models etc.
- Plus, **software is usually only part of a larger system**
- And it’s the **system** we care about

John Rushby, SRI
Software Correctness vs. System Claims

- The top-level requirements for most complex systems are stated quantitatively
  - E.g., no catastrophic failure in the lifetime of all airplanes of one type

- And these lead to probabilistic requirements for software-intensive subsystems
  - E.g., probability of failure in flight control less than $10^{-9}$ per hour

- But formal methods in general
  - And formal verification in particular
    Are about correctness... an absolute notion

- How do we connect the absolute claims of formal verification for software with probabilistic requirements at the system level?

John Rushby, SRI
Software Reliability

- Software contributes to system failures through faults in its requirements, design, implementation—bugs

- A bug that leads to failure is certain to do so whenever it is encountered in similar circumstances
  - There’s nothing probabilistic about it

- Aaah, but the circumstances of the system are a stochastic process

- So there is a probability of encountering the circumstances that activate the bug

- Hence, probabilistic statements about software reliability or failure are perfectly reasonable

- Typically speak of probability of failure on demand (pfd), or failure rate (per hour, say)
Measuring/Predicting Software Reliability

- For pfds down to about $10^{-4}$, it is feasible to measure software reliability by statistically valid random testing.
- But $10^{-9}$ would need 114,000 years on test.
- So how do we establish that a piece of software is adequately reliable for a system that requires, say, $10^{-6}$?
- Most standards for system safety (e.g., IEC 61508, DO178B) require you to show that you did a lot of V&V:
  - e.g., 57 V&V “objectives” at DO178B Level C ($10^{-5}$)
- And you have to do more for higher levels:
  - 65 objectives at DO178B Level B ($10^{-7}$)
  - 66 objectives at DO178B Level A ($10^{-9}$)
Does “More Correct” Mean More Reliable?

● These V&V objectives are all about correctness
  ○ Requirements tracing, testing etc.

● More V&V objectives might make the software “more correct” but what does that have to do with reliability?

● And what does “more correct” mean anyway?
Possibly Perfect Software

- Instead of correct software
  - Which is about conformance with specification
- We’ll speak of perfect software
  - Software that will never experience a failure in operation, no matter how much operational exposure it has
- You might not believe a given piece of software is perfect
- But you might concede it has a possibility of being perfect
- And the more V&V it has had, the greater that possibility
- So let’s speak of a probability of perfection
  - Think of all the software that might have been developed by comparable engineering processes to solve the same design problem as the software at hand
  - The probability of perfection is then the probability that any software randomly selected from this class is perfect
Probabilities of Perfection and Failure

- Probability of perfection relates to correctness-based V&V
- And it also relates to reliability:

By the formula for total probability

\[
P(s/w \text{ fails [on a randomly selected demand]}) = P(s/w \text{ fails } | s/w \text{ perfect}) \times P(s/w \text{ perfect}) \]
\[
+ P(s/w \text{ fails } | s/w \text{ imperfect}) \times P(s/w \text{ imperfect}).
\]

- The first term in this sum is zero, because the software does not fail if it is perfect
- Can then, very conservatively, assume that the software always fails if it imperfect, so that the first factor in the second term becomes 1

Hence, very crudely, and very conservatively,

\[
P(\text{software fails}) < P(\text{software imperfect})
\]

John Rushby, SRI
Two Channel Systems

- Many safety-critical systems have two (or more) diverse “channels”
  - E.g., nuclear shutdown, flight control
- One operational channel does the business
- A simpler channel provides a backup or monitor
- Cannot simply multiply the pfds of the two channels to get pfd for the system
  - Failures are unlikely to be independent
  - Failure of one channel suggests this is a difficult case, so failure of the other is more likely
  - Infeasible to measure amount of dependence
Two Channel Systems and Possible Perfection

- But if the second channel is possibly perfect
  - Its imperfection is conditionally independent of failures in the first channel
  - Hence, system pfd is conservatively bounded by product of pfd of first channel and probability of imperfection of the second

- Joint work with Bev Littlewood (reported elsewhere)
  - Who originated the idea of possible perfection

- May provide justification for some of the architectures suggested in ARP 4754
  - e.g., $10^{-9}$ system made of Level C operational channel and Level A monitor
Aside: Monitors Do Fail

- Fuel emergency on Airbus A340-642, G-VATL, 8 February 2005
  - Type 1 failure

- EFIS Reboot during spin recovery on Airbus A300 (American Airlines Flight 903), 12 May 1997
  - Type 2 failure

- Current proposals are for formally synthesized/verified monitors for properties in the safety case
Aleatory and Epistemic Uncertainty

- **Aleatory** or irreducible uncertainty
  - is “uncertainty in the world”
  - e.g., if I have a biased coin with $P(\text{heads}) = p_h$, I cannot predict exactly how many heads will occur in 100 trials because of randomness in the world
    - Frequentist interpretation of probability needed here

- **Epistemic** or reducible uncertainty
  - is “uncertainty about the world”
  - e.g., if I give you the biased coin, you will not know $p_h$; you can estimate it, and can try to improve your estimate by doing experiments, learning something about its manufacture, the historical record of similar coins etc.
    - Frequentist and subjective interpretations OK here
Aleatory and Epistemic Uncertainty in Models

- In much scientific modeling, the **aleatory** uncertainty is captured conditionally in a model with parameters.

- And the **epistemic** uncertainty centers upon the **values of these parameters**.

- As in the coin tossing example.

- Analysis in (1) was **aleatory**, with parameters:
  - $p_{np}$ probability the software is imperfect
  - $p_{fnp}$ probability that it fails, if it is imperfect
  - $P($software fails$) < p_{fnp} \times p_{np}$
Epistemic Estimation

- To apply this result, we need to assess values for $p_{fn}$ and $p_{np}$.
- These are most likely subjective probabilities:
  - i.e., degrees of belief
- Beliefs may not be independent.
- So will be represented by some joint distribution $F(p_{fn}, p_{np})$.
- Probability of system failure will be given by the Riemann-Stieltjes integral
  \[
  \int_{0 \leq p_{fn} \leq 1} \int_{0 \leq p_{np} \leq 1} p_{fn} \times p_{np} \, dF(p_{fn}, p_{np}).
  \]
  (3)
- If beliefs can be separated $F$ factorizes as $F(p_{fn}) \times F(p_{np})$.
- And (3) becomes $P_{fn} \times P_{np}$.
  Where these are the means of the posterior distributions representing the assessor’s beliefs about the two parameters.
Formal Verification and the Probability of Perfection

- We want to assess $P_{np}$

- Context is likely a safety case in which claims about a system are justified by an argument based on evidence about the system and its development

- Suppose part of the evidence is formal verification
  - i.e., what is the probability of perfection of formally verified software?

- Let’s consider where formal verification can go wrong
The Basic Requirements For The Software Are Wrong

- This error is made before any formalization
- It seems to be the dominant source of errors in flight software
- But monitoring and backup software is built to requirements taken directly from the safety case
  - If these are wrong, we have big problems
- In any case, it’s not specific to formal verification
- So we’ll discount this concern
The Requirements etc. are Formalized Incorrectly

- Could also be the assumptions, or the design

- Formalization may be inconsistent
  - i.e., meaningless
  - Many Z specs are like this

Can be eliminated using constructive specifications
  - In a tool-supported framework
  - That guarantees conservative extension

But that’s not always appropriate
  - Prefer to state assumptions as axioms
  - Consistency can then be guaranteed by exhibiting a constructive model (interpretation)
  - PVS can do this

- So we can eliminate this concern
The Requirements etc. are Formalized Incorrectly (ctd.)

- Formalization may be consistent, but **wrong**
- **In my experience, this is the dominant source of errors in formal verification**
  - There are papers on errors in my specifications
- Formal specifications that have not been subjected to analysis are no more likely to be correct than programs that have never been run
  - In fact, less so: engineers have better intuitions about programs than specifications
- Should **challenge** formal specifications
  - Prove putative theorems
  - Get counterexamples for deliberately false conjectures
  - Directly execute them on test cases
- **Social process operates on widely used theories**
The Requirements etc. are Formalized Incorrectly (ctd. 2)

• Even if a theory or specification is formalized incorrectly, it does not necessarily invalidate all theorems that use it

• Only if the verification actually exploits the incorrectness will the validity of the theorem be in doubt
  ◦ Even then, it could still be true, but unproven

• Some verification systems identify all the axioms and definitions on which a formally verified conclusion depends
  ◦ PVS does this

If these are correct, then logical validity of the verified conclusion follows by soundness of the verification system
  ◦ **So can apply special scrutiny to them**
The Formal Specification and Verification is Discontinuous or Incomplete

- **Discontinuities** arise when several analysis tools are applied in the same specification
  - e.g., static analyzer, model checker, timing analyzer
  Concern is that different tools ascribe different semantics

- Increasing issue as specialized tools outstrip monolithic ones
  - Need integrating frameworks such as a tool bus

- Most significant **incompleteness** is generally the gap between the most detailed model and the real thing
  - Algorithms vs. code, libraries, OS calls

That's one reason why we still need testing
  - Driven from the formal specification
  - Cf. penetration tests for secure systems: probe the assumptions
Unsoundness In the Verification System

- All verification systems have had soundness bugs
- But none have been exploited to prove a false theorem
- Many efforts to guarantee soundness are costly
  - e.g., reduction to elementary steps, proof objects
  - What does soundness matter if you cannot do the proof?
- A better approach is KOT: the Kernel Of Truth (Shankar)
  - A ladder of increasingly powerful verified checkers
  - Untrusted prover leaves a trail, blessed by verified checker
  - More powerful checkers guaranteed by one-time check of its verification by the one above
  - The more powerful the verified checker, the more economical the trail can be (little more than hints)
Shankar and Marc Vaucher have verified a modern SAT solver that is executable (modulo lacunae in the PVS evaluator)
Application

- Suppose we need $P_{np}$ of $10^{-4}$
- Bulk of this “budget” should be divided between incorrect formalization and incompleteness of the formal analysis, with small fraction allocated to unsoundness of verification system
- Through sufficiently careful and comprehensive formal challenges, it is plausible an assessor can assign a subjective posterior probability of imperfection on the order of $10^{-4}$ to the formal statements on which a formal verification depends
- Through testing and other scrutiny, a similar figure can be assigned to the probability of imperfection due to discontinuities and incompleteness in the formal analysis
- By use of a verification system with a trusted or verified kernel, or trusted, verified, or diverse checkers, assessor can assign probability of $10^{-5}$ or smaller that the theorem prover incorrectly verified the theorems that attest to perfection
Discussion

• These numbers are feasible and plausible

• Formal methods and their tools do not need to be held to (much) higher standards than the systems they assure

• But what are we to do about single channel systems that require $10^{-9}$?
  ○ Type 2 failures of monitors (incorrect activation) may be in this class
  ○ Topic for investigation and discussion whether such assessments could be considered feasible and credible
Conclusion

• **Probability of perfection** is a radical and valuable idea

• **Provides the bridge between correctness-based verification activities and probabilistic claims needed at the system level**

• **Relieves formal verification, and its tools, of the burden of absolute perfection**