Marktoberdorf NATO Summer School 2016, Lecture 1

Assurance and Formal Methods

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, CA

Requirements, Assumptions, Specifications

- There is an environment, aka. the world (given)
- And a system (to be constructed)
- Assumptions A describe behavior/attributes of the environment that are true independently of the system
    - Expressed entirely in terms of environment variables
- Requirements R describe desired behavior in the environment
    - Expressed entirely in terms of environment variables
- There's a boundary/interface between system & environment
    - Typically shared variables (e.g., 4-variable model)
- Specification S describes desired behavior on shared variables
- Correctness is $A, S \vdash R$ and $A, I \vdash S$, where I is implementation

The Fault, Error, Failure Chain

Failure: departure from requirements

For critical failures, the requirement is sometimes implicit

Error: discrepancy between actual and intended behavior (inside system boundary)

Fault: a defect (bug) in a system

- Faults (may) cause errors, which (may) cause failure
  - What about errors not caused by a fault, such as bit-flips caused by alpha particles?
  - These are environmental phenomena, should appear in assumptions, requirements; fault is not dealing with them
- Failure in a subsystem (may) cause an error in the system
- Fault tolerance is about detecting and repairing or masking errors before they lead to failure
- Formal methods is typically about detecting faults
- Verification is about guaranteeing absence of faults

Critical Failures

- System failures can cause harm
    - To people, nations, the world
- Harm can occur in many dimensions
    - Death and injury, theft and loss (of property, privacy), loss of service, reduced quality of life
- I will mostly focus on critical failures
    - Those that do really serious harm
- Serious faults are often in the requirements
    - $A, S \vdash$ violation of implicit requirements due to
    - $A, R \vdash$ violation of implicit requirements
    - But for this lecture we'll assume requirements are OK
- Generally want severity of harm and frequency of occurrence to be inversely related
- Risk is the product of severity and frequency

Risk

- Public perception and tolerance of risk is not easy to explain
  - Unrelated to statistical threat, mainly "dread factor":
    involuntary exposure, uncontrollable, mass impact
- US data, annual deaths (typical recent years)
  - Medical errors: 440,000
  - Road accidents: 35,000
  - Firearms: 12,000
    mass shootings [≥ 4 victims]: more than 1 a day
    (but other crime is quite low in the US)
  - Terrorism: 30
  - Plane crashes: 0
  - Train crashes: 0
  - Nuclear accidents: 0
- UK data: cyber crime (2.11m victims) exceeds physical crime
- Our task is to ensure low risk for computerized systems

Assurance Requirements

- For a given severity of harm, we need to guarantee some acceptable upper bound on frequency of failure
- Example: aircraft failure conditions are classified in terms of the severity of their consequences
- Catastrophic failure conditions are those that could prevent continued safe flight and landing
- And so on through severe major, major, minor, to no effect
- Severity and probability/frequency must be inversely related
- AC 25.1309: No catastrophic failure conditions expected to occur in the operational life of all aircraft of one type
- Arithmetic, history, and regulation require the probability of catastrophic failure to be less than 10^{-9} per hour, sustained for many hours
- Similar for other critical systems and properties

Software Assurance and Software Reliability

- Software contributes to system failures through faults in its specifications, design, implementation—bugs
- Assurance requirements are expressed in terms of probabilities
- But a fault 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 fault and lead to failure
- 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)

Assurance in Practice

- Prior to deployment, the only direct way to validate a reliability requirement (i.e., rate or frequency of failure) is by statistically valid random testing
    - Tests must reproduce the operational profile
    - Requires a lot of tests
        - ⋆ Must not see any failures
    - Infeasible to get beyond 10^{-3}, maybe 10^{-4}
    - 10^{-9} is completely out of reach
- Instead, most assurance is accomplished by coverage-based testing, inspections/walkthroughs, formal methods
- But these do not measure failure rates
- They attempt to demonstrate absence of faults
- So how is absence of faults related to frequency of failure?
- Let's focus on formal verification

Formal Verification and Assurance

- Suppose we formally verify some property of the system
- This guarantees absence of faults (wrt. those properties)
- Guarantees?
  - Suppose theorem prover/model checker is unsound?
  - Or assumed semantics of language is incorrect?
  - Or verified property doesn't mean what we think it means?
  - Or environment assumptions are formalized wrongly?
  - Or ancillary theories are formalized incorrectly?
  - Or we model only part of the problem, or an abstraction?
  - Or the requirements were wrong?
- Must admit there's a possibility the verification is incorrect
  - Or incomplete
- How can we express this?
- As a probability!

Probability of Fault-Freeness

- Verification and other assurance activities aim to show the software is free of faults
- The more assurance we do, the more confident we will be in its fault-freeness
- Can express this confidence as a subjective probability that the software is fault-free or <u>n</u>on<u>f</u>aulty: p_{nf}
    - Or perfect: some papers speak of probability of perfection
- For a frequentist interpretation: think of all the software that might have been developed by comparable engineering processes to solve the same design problem
    - And that has had the same degree of assurance
    - Then p_{nf} is the probability that any software randomly selected from this class is nonfaulty
- Fault-free software will never experience a failure, no matter how much operational exposure it has

Relationship Between Fault-Freeness and Reliability

- By the formula for total probability

$$
\begin{aligned}
& P(\text{s/w fails [on a randomly selected demand]}) \qquad (1)\\
= \; & P(\text{s/w fails} \,|\, \text{s/w fault-free}) \times P(\text{s/w fault-free})\\
& + P(\text{s/w fails} \,|\, \text{s/w faulty}) \times P(\text{s/w faulty}).
\end{aligned}
$$

- The first term in this sum is zero
  - Because the software does not fail if it is fault-free
  - Which is why the theory needs this property
- Define $p_{F|f}$ as the probability that it Fails, if faulty
- Then (1) becomes $pfd = p_{F|f} \times (1 - p_{nf})$

Aleatoric and Epistemic Uncertainty

- Aleatoric or irreducible uncertainty
  - is “uncertainty in the world”
  - e.g., if I have a coin with $P(heads) = p_h$, 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 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

Aleatoric and Epistemic Uncertainty in Models

- In much scientific modeling, the aleatoric uncertainty is captured conditionally in a model with parameters
- And the epistemic uncertainty centers upon the values of these parameters
- In the coin tossing example: p_h is the parameter
- In our software assurance model

$$pfd = p_{F|f} \times (1 - p_{nf})$$

$p_{F|f}$ and p_{nf} are the parameters

Epistemic Estimation

- To apply our model, we need to assess values for $p_{F|f}$ and p_{nf}
- These are most likely subjective probabilities
  - i.e., degrees of belief
- Beliefs about $p_{F|f}$ and p_{nf} might not be independent
- So will be represented by some joint distribution $F(p_{F|f}, p_{nf})$
- Probability of software failure will be given by the Riemann-Stieltjes integral

$$\int_{\substack{0 \le p_{F|f} \le 1 \\ 0 \le p_{nf} \le 1}} p_{F|f} \times (1 - p_{nf})\, dF(p_{F|f}, p_{nf}) \qquad (2)$$

- If beliefs can be separated F factorizes as $F(p_{F|f}) \times F(p_{nf})$
- And (2) becomes $P_{F|f} \times (1 - P_{nf})$

  Where $P_{F|f}$ and P_{nf} are means of the posterior distributions representing the assessor's beliefs about the two parameters
- One way to separate beliefs is via conservative assumptions

Practical Application—Nuclear

- Traditionally, UK nuclear protection systems are assured by statistically valid random testing
- Very expensive to get required pfd of 10^{-4} this way
- Our analysis says pfd $\leq P_{F|f} \times (1 - P_{nf})$
- They are essentially setting P_{nf} to 0 and doing the work to assess $P_{F|f} < 10^{-4}$
- Any assurance process that could give them $P_{nf} > 0$
- Would reduce the amount of testing they need to do
  - e.g., $P_{nf} > 1 - 10^{-1}$, which seems very plausible
  - Would deliver the same pfd with $P_{F|f} < 10^{-3}$
- This could reduce the total cost of assurance and certification

Practical Application—Aircraft, Version 1

- Aircraft software is assured by V&V processes such as ARP-4754A, and DO-178C Level A
- Need software failure rate $< 10^{-9}$
- As well as DO-178C, they also do a massive amount of all-up testing but do not take assurance credit for this
- Our analysis says software failure rate $\leq P_{F|f} \times (1 - P_{nf})$
- So they are setting $P_{F|f} = 1$ and $P_{nf} > 1 - 10^{-9}$
- This is completely implausible as an a priori assessment
- Even if they implicitly get $P_{F|f} \leq 10^{-3}$ from testing, they still would need $P_{nf} > 1 - 10^{-6}$
- Which is also implausible
- There must be another explanation

Relationship Between Fault-Freeness and Survival

- Instead of failure on individual demands, look at survival over many
- The probability $p_{srv}(n)$ of surviving n independent demands (e.g., flights) without failure is given by

$$p_{srv}(n) = p_{nf} + (1 - p_{nf}) \times (1 - p_{F|f})^n \tag{3}$$

- A suitably large n can represent "the entire lifetime of all aircraft of one type"
  - 2,000 planes × 25 years × 5.5 flights per day gives $n = 10^8$
- First term in (3) establishes a lower bound for $p_{srv}(n)$ that is independent of n
- If assurance gives us the confidence to assess, say, $p_{nf} > 0.9$
- Then we are almost there
- Just need some contribution from the second term

Practical Application—Aircraft, Version 2

- We need confidence that the second term in (3) will be nonzero, despite exponential decay
- Confidence could come from prior failure-free operation
- Calculating overall $p_{srv}(n)$ is a problem in Bayesian inference
  - We have assessed a value for P_{nf}
  - Have observed some number r of failure-free demands
  - Want to predict prob. of $n - r$ future failure-free demands
- Need a prior distribution for $P_{F|f}$
  - Difficult to obtain, and difficult to justify for certification
  - However, there is a distribution that delivers provably worst-case predictions
    - ⋆ One where $P_{F|f}$ is a probability mass at some $q_n \in (0, 1]$
  - So can make predictions that are guaranteed conservative, given only P_{nf}, r, and n

Practical Application—Aircraft, Version 2 Continued

- For values of p_{nf} above 0.9
- The second term in (3) is well above zero
- Provided $r > \frac{n}{10}$
- So it looks like we need to fly 10^7 hours to certify 10^8
- Maybe not!
- Entering service, we have only a few planes, need confidence for only, say, first six months of operation, so a small n
- Flight tests are enough for this
- Next six months, have more planes, but can base prediction on first six months (or ground the fleet, fix things)
- And bootstrap our way forward
- This is a rational reconstruction of how aircraft software certification could work (due to Strigini and Povyakalo)
- It provides a model that is consistent with practice

Why This Matters

- We don't really know how/why certification works
  - And it does seem to work
- And we don't really know what makes for effective standards/guidelines
- But we need to make changes
  - New kinds of systems
  - New methods of software development
  - New methods of analysis/verification
  - Desire to reduce costs
- Now we know it comes down to assessing useful values for p_{nf}
  - i.e., effective methods and tools for analysis
    - ⋆ That's for software, we don't have much for systems
  - And coherent treatment for all the attendant doubts

Variant: Monitoring

- In some systems, it's feasible to have a simple monitor that can shut off a more complex operational component
  - Turns malfunction and unintended function into loss of function
  - Prevents transitions into unsafe states
- Reliability of the whole is not the product of the reliabilities of the operational and monitor components
- But it is a theorem that the fault freeness of the monitor is independent of the reliability of the operational component
- And reliability of the whole is the product of these
  - At aleatoric level, it's more complex for epistemic
  - Must also deal with undesired monitor activation
- Application (also known as runtime verification)
  - Formally synthesize monitor from formal safety constraints
  - Feasible to assess good p_{nf} for the monitor
- Significant overall benefit at relatively low cost

Monitoring Example: A340 fuel management

- Fuel emergency on Airbus A340-642, G-VATL, on 8 February 2005 (AAIB SPECIAL Bulletin S1/2005)
- Toward the end of a flight from Hong Kong to London: two engines flamed out, crew found certain tanks were critically low on fuel, declared an emergency, landed at Amsterdam
- Two Fuel Control Monitoring Computers (FCMCs) on this type of airplane; each a self-checking pair with a backup (so 6-fold redundant in total); they cross-compare and the "healthiest" one drives the outputs to the data bus
- Both FCMCs had fault indications, and one of them was unable to drive the data bus
- Unfortunately, this one was judged the healthiest and was given control of the bus even though it could not exercise it
- The backups were suppressed because the FCMCs indicated they were not both failed
- Contemplate a monitor synthesized from the safety requirements

Coming Up

Next, we'll look at how assurance can justify a claim such as $p_{nf} > 0.9$

References

[1] Bev Littlewood and John Rushby. Reasoning about the reliability of diverse two-channel systems in which one channel is "possibly perfect". *IEEE Transactions on Software Engineering*, 38(5):1178–1194, September/October 2012.

[2] Lorenzo Strigini and Andrey Povyakalo. Software fault-freeness and reliability predictions. In SafeComp *2013: Proceedings of the 32nd International Conference on Computer Safety, Reliability, and Security*, Volume 8153 of Springer-Verlag *Lecture Notes in Computer Science*, pages 106–117, Toulouse, France, September 2013.