What is Formal Verification, Why is its Importance Increasing, and How is it Developing?

John Franco
Computer Science, University of Cincinnati
What is Formal Verification?
What is Formal Verification?

For X in Hardware, Software, Protocols, Systems:

- Prove that X does what it is supposed to do
- and nothing more!
What is Formal Verification?

For X in Hardware, Software, Protocols, Systems:

• Prove that X does what it is supposed to do
• and nothing more!

The item of the second bullet is harder than that of the first

• You have to enumerate all things that can go wrong but most people, even groups, cannot make a complete list
• To prove that X does not have a property is usually much harder than to prove X has it
Why Isn’t Testing Enough?

• The major goal of software testing is to discover errors in the software with a secondary goal of building confidence in the proper operation of the software when testing does not discover errors.
Why Isn’t Testing Enough?

- The major goal of software testing is to discover errors in the software with a secondary goal of building confidence in the proper operation of the software when testing does not discover errors.
- In the absence of other information, this could mean either that the software is high quality or that the testing process is low quality.
Why Isn’t Testing Enough?

- The major goal of software testing is to discover errors in the software with a secondary goal of building confidence in the proper operation of the software when testing does not discover errors.

- In the absence of other information, this could mean either that the software is high quality or that the testing process is low quality.

- Program testing can be used to show the presence of bugs, but never to show their absence - Edsger Dijkstra
Some Definitions

Security (IEEE):
The protection of computer hardware and software from accidental or malicious access, use, modification, destruction, or disclosure.
Some Definitions

Safety (DoD):
Freedom from those conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property, or damage to the environment.
Some Definitions

Safety critical (DoD):
A term applied to a condition, event, operation, process or item of whose proper recognition, control, performance or tolerance is essential to safe system operation or use; e.g. safety critical function, safety critical path, safety critical component.
Some Definitions

**Safety critical computer software components (DoD):**

Those computer software components and units whose errors can result in a potential hazard, or loss of predictability or control of a system.
Requirements vs. Specification

Requirements:
Statement of what the customer wants and needs.

Specifications:
How customer requirements are met by the system design.
Some Definitions

Requirements (IEEE):

- A condition or capability needed by a user to solve a problem or achieve an objective.
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
- A documented representation of a condition or capability as in 1. or 2. above.
Some Definitions

Specification (IEEE):

A document that specifies, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system or component, and often, the procedures for determining whether these provisions have been satisfied.
Some Definitions

Specification analysis (IEEE):

Evaluation of each safety-critical software requirement with respect to a list of qualities such as completeness, correctness, integrity, consistency, testability, robustness, reliability, usability, flexibility, maintainability, portability, interoperability, accuracy, auditability, performance, internal instrumentation, security and training.
Some Definitions

Design specification (NIST):
A specification that documents how a system is to be built. It typically includes system or component structure, algorithms, control logic, data structures, data set use information, input/output formats, interface descriptions, etc.
Some Definitions

Design standards (IEEE):

Standards that describe the characteristics of a design or a design description of data or program components.
Some Definitions

Formal specification (NIST):

- A specification written and approved in accordance with established standards.
- A specification expressed in a requirements specification language.
Some Definitions

**Functional specification (NIST):**

A specification that documents the functional requirements for a system or system component. It describes what the system or component is to do rather than how it is to be built. Often part of a requirements specification.
Some Definitions

Interface specification (NIST):

A specification that documents the interface requirements for a system or system component. Often part of a requirements specification.
Some Definitions

Performance specification (IEEE):
A document that sets forth the performance characteristics that a system or component must possess. These characteristics typically include speed, accuracy, and memory usage. Often part of a requirements specification.
Some Definitions

Product specification (IEEE):

A document which describes the as-built version of the software
Some Definitions

Requirements specification (NIST):

A specification that documents the requirements of a system or system component. It typically includes functional requirements, performance requirements, interface requirements design requirements, development standards, etc.
**Some Definitions**

**Type Safety:**

A language is type-safe if the only operations that can be performed on data in the language are those sanctioned by the type of the data.

Code is memory type-safe if only authorized memory locations are accessed.
Some Definitions

Java is not type safe:

In the end they sacrificed safety for power.
The problem is with class loaders and reflection.
But functional languages Haskell, ML are type safe.
Requirements vs. Specification

Requirements:
Statement of what the customer wants and needs.
Used for validation.

Specifications:
How customer requirements are met by the system design.
Used for verification.
Validation vs. Verification

Validation (Quality Assurance):
   Does the device meet the customer’s requirements?
   Have we built the correct device?

Verification (Quality Control):
   Have we built the device correctly?
   Does the device meet specifications?
   Did we find and remove all of the 'bugs'?
<table>
<thead>
<tr>
<th>property</th>
<th>C/C++</th>
<th>Java</th>
<th>Lisp/ML/Haskell</th>
</tr>
</thead>
<tbody>
<tr>
<td>speed</td>
<td>Fast</td>
<td>Middle</td>
<td>Slow</td>
</tr>
<tr>
<td>security</td>
<td>None</td>
<td>Substantial</td>
<td>Substantial, no side effects</td>
</tr>
<tr>
<td>safety</td>
<td>None</td>
<td>Little</td>
<td>Slight</td>
</tr>
<tr>
<td>type safe</td>
<td>No</td>
<td>No</td>
<td>ML, Haskell-Yes</td>
</tr>
<tr>
<td>prototype</td>
<td>Slow</td>
<td>Moderate</td>
<td>Fast</td>
</tr>
<tr>
<td>errors</td>
<td>Many</td>
<td>Some</td>
<td>Some</td>
</tr>
<tr>
<td>maintainable</td>
<td>Difficult</td>
<td>Moderate</td>
<td>Most easily maintainable</td>
</tr>
<tr>
<td>checking</td>
<td>Runtime</td>
<td>Compile time</td>
<td>Runtime/Compile time</td>
</tr>
</tbody>
</table>
So Good Tools Are Needed

Model Checking:
- SAT Solvers, BDDs, Satisfiability Modulo Theories.
- Temporal Logics.

Equivalence Checking:
- SAT Solvers.
- Cryptol.

Theorem Proving:
- ACL2, Temporal Logics.

Correct by Construction:
- Kestrel? Future?
Input Representations for These Tools

Model Checking:
- BDDs; CNF, QF, Quantified propositional formulas.

Equivalence Checking:
- And-Inverter Graphs.

Theorem Proving:
- First order logic.
What Can Be Verified?

- A formal model obtained from a formal specification is equivalent to a formal model obtained for a software or hardware implementation (correctness).
- Unauthorized access to cryptovariables cannot happen.
- The possibility of read-before-write is nil.
- The possibility of return address overwrite is nil.
Modes of Validation/Verification

A state-space:
- Initial state, transitions between states, paths of execution.

Concrete Execution:
- Execution of a path from the initial state.
  Program or circuit values are tracked.

Symbolic Evaluation:
- Many paths in state-space are explored simultaneously.
  Symbolic variables are tracked instead of values.

Concolic Testing:
- Hybrid - intereaves concrete execution with symbolic execution.