Dijkstra and De Millo
Challenges in Applying and Teaching Verification

Zach Tatlock and Xi Wang
with numerous colleagues
When exhaustive testing is impossible, our trust can only be based on proof.

Edsger W. Dijkstra
*Under the Spell of Leibniz’s Dream*

Verification will never work!

Four decades pass ...
Victory?
Challenging to apply to existing systems

Few verification experts in the field
UW Verification Projects and Courses

- Radiotherapy Device
- Distributed Systems
- Filesystems
- Memory Models
- Compilers

- Systems Verification
- Constraint Solvers
- Proof Assistants
- Hack Your Language
- NLP + PL

...
UW Verification Projects and Courses

- Radiotherapy Device
- Distributed Systems
- Filesystems
- Memory Models
- Compilers

- Systems Verification
- Constraint Solvers
- Proof Assistants
- Hack Your Language
- NLP + PL

...
Formal

End-to-end

Safety Cases

COMPCERT
IronFleet
FSCQ
Chapar
CertikOS
Mem Model
IronClad
JitK
Verdi
Quark
Vellvm
CertiCrypt
seL4
miTLS
Safety Cases

Integrate diverse source of evidence

*check interfaces of design, testing, proof, review*

Argue end-to-end claim based on evidence

*show claim holds across all layers of a system*

Focus on physical system properties

*eases validation and focuses verification effort*
Many large software systems display fragility or a lack of dependability caused by inattention to details at various stages of development (e.g., missing data, undocumented assumptions, lack of testing), resulting in a failure to catch errors. This technical note explains how to create a dependability case for a system that helps identify and keep track of such details.

A dependability case is defined here as a structured argument providing evidence that a system meets its specified dependability requirements. The technical note describes how to structure the argument and present evidence to support it. A sample problem is presented, as well as issues raised by that problem and future goals. Many large software systems display fragility or a lack of dependability caused by inattention to details at various stages of development (e.g., missing data, undocumented assumptions, lack of testing), resulting in a failure to catch errors. This technical note explains how to...
Formal End-to-end

Checkable Cases

Safety Cases

- COMPCERT
- IronFleet
- Mem Model
- FSCQ
- Chapar
- IronClad
- Verdi
- CertiKOS
- JitK
- Vellvm
- CertiCrypt
- Quark
- seL4
- miTLS

frenetic
Example Safety Case for CNTS
Example Safety Case for CNTS

Prescription Safety:
If any setting exceed prescribed tolerances, the beam will turn off and stay off.
Example Safety Case for CNTS

rotation $\notin$ tolerances $\rightarrow$ beam off
Example Safety Case for CNTS

rotation $\not\in$ tolerances $\rightarrow$ beam off

rotation changes $\rightarrow$ transmits reading

receives bad reading $\rightarrow$ triggers interlock

interlock triggered $\rightarrow$ beam off
Example Safety Case for CNTS

rotation \not\in \text{tolerances} \rightarrow \text{beam off}

rotation changes \rightarrow \text{transmits reading}

receives bad reading \rightarrow \text{triggers interlock}

interlock triggered \rightarrow \text{beam off}

- Sensors + Controller
- Therapy Control
- PLC + HSIS
Example Safety Case for CNTS

No support for automated checking

No links from claims to implementation
Example **Checkable** Safety Case for CNTS

**System Model**

- **Sensors + Controller**
- **Therapy Control**
- **PLC + HSIS**

**Checkers**
Example **Checkable** Safety Case for CNTS

**System Model**

- Rotation not in tolerances → beam off
- Rotation change → random misreading
- Receives bad reading, triggers interlock
- Interlock triggers beam off

**Checkers**

- Sensors + Controller
- Therapy Control
- PLC + HSIS
Example **Checkable** Safety Case for CNTS

System model in Alloy

*specifies state space*

*formalizes safety case*

Checker plug-in architecture

*connect claims to evidence*

*specialized to implementation*
Example **Checkable** Safety Case for CNTS

- Automatically check safety properties
- Incorporate evidence from implementation
- Simplify checker development
Several Issues Identified

- Beam may fail to shut off
  - *bad gantry rotation check*
  - *array indexing discrepancy*

- Unreported error conditions
  - *broken dataflow links*

- Inaccurate system model
  - *missing PLC relay*
Gantry Rotation Bug

EPICS Control Software Checker:

(("Gantry:Rotation:Prescribed" 315)
("Gantry:Rotation:Actual" 45 ...))

angles mod 360

gantry_couch.subst:29

UW Verification Projects and Courses

- Radiotherapy Device
- Distributed Systems
- Filesystems
- Memory Models
- Compilers
- Systems Verification
- Constraint Solvers
- Proof Assistants
- Hack Your Language
- NLP + PL
599W: Systems Verification

• Graduate course at U. Washington, spring 2016
• Instructors: Bryan Parno, Zach Tatlock, Xi Wang
• Reflections based on students feedback
  • Toolchain
  • End-to-end correctness
  • Proof automation
• Disclaimer
Class structure

• 3 weeks: crash course on verification tools
  • Z3/Rosette, Dafny, Coq
  • All the students had taken a grad PL class in Coq

• 7 weeks: paper discussions
  • Instructor-led: CompCert, IronFleet
  • Student-led: Verdi, FSCQ, seL4, RockSalt, HMAC, Ironclad, miTLS
    • Guest lecture: Gernot Heiser

• Class project
Student background (17 in total)

- PL: 6
- Sys/Net: 5
- Misc: 4
- Security: 2
Tools in class projects (11 in total)

- Coq, 5
- SMT, 4
- N/A, 3
- TLA, 1
### Tools by student background

<table>
<thead>
<tr>
<th></th>
<th>Coq</th>
<th>SMT</th>
<th>TLA</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL</td>
<td>5</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Sys/Net</td>
<td>-</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Security</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>Misc.</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>9</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Summary of toolchain 1/2

• 1st choice: Coq
  • 9 PL+Misc students; none outside the PLSE group
  • Pros: grad PL class, books, community support

• 2nd choice: SMT (Z3 and Rosette)
  • 1 PL and 3 Sys/Net students
  • Pros: grad PL class, automation, synthesis
Summary of toolchain 2/2

• Why not Dafny, Isabelle, Lean, etc.
• Reasons given by students
  • Lack of in-house expertise
  • Existing projects are in Coq/SMT
  • Need to know the whole stack (Boogie/Z3 & triggers!)
  • IDE and platform dependency
• Social factors rather than technical reasons
Students feedback on papers

• End-to-end correctness of verified systems
  • “What’s the empirical evidence that they’re better”
  • “What does formal guarantee mean for security”
  • “Secure/non-secure vs hardening”

• Proof burden & automation
  • “The proof burden is just too high”
  • “Almost every paper claims proof automation”
Ongoing: empirical study

• Inspired by Csmith [PLDI’11] on compiler testing
• Verified distributed systems
  • IronFleet (Dafny), Verdi (Coq), Chapar (Coq)
  • Fuzz testing, code review, bug injection
• Control group: unverified production systems
  • LogCabin, ZooKeeper, etcd, Cassandra
  • Categorize bug reports from the past 12 months
• Threats to validity
Current results

• Are verified distributed systems better?
  • Short answer: yes
  • No bugs found in verified parts given correct specs
  • No protocol bugs found - not because of lack of trying

• What can go wrong in verified distributed systems?
  • 16 bugs found in tools, specs, runtime library; 4 fixed
  • Example: tool always prints “verified”
  • Example: bug in OCaml library Marshal.to_channel()

• Can verification prevent bugs in production sys.?
Ongoing: push-button verification

- Co-design a file system with verification
- Goal: no proofs
  - Programmers write spec, impl, fsck invariants
  - No loop invariants nor annotations on code
  - Debugging friendly: able to produce counterexamples
- Fully automated SMT reasoning
  - Can be considered as exhaustive symbolic execution
  - Achieve scalability by layered composition
  - Get rid of unbounded loops via translation validation
Current results

• Yxv6: a verified journaling file system
  • Similar to the xv6 file system and FSCQ
  • Written in Python/Z3 & compiled to C for execution
  • No low-level bugs & all paths exhausted
  • Functional correctness & crash safety

• Push-button verification for Yxv6
  • \(\sim300\) LOC spec and \(\sim3,000\) LOC Implementation
  • Little proof burden: 5 fsck invariants
  • Easy to optimize & verify
  • < 3 months to achieve self-hosting development
UW Verification Projects and Courses

- Radiotherapy Device
- Distributed Systems
- Filesystems
- Memory Models
- Compilers

- Systems Verification
- Constraint Solvers
- Proof Assistants
- Hack Your Language
- NLP + PL

...