When Formal System Kill: Computer Ethics and Formal Methods

Lee Pike (presenting)[1] Darren Abramson[2]

[1]Galois Inc.
leepike@galois.com

[2]Department of Philosophy,
Dalhousie University

July 27, 2007
North American Computers and Philosophy (NA-CAP) Conference

Brief biographical aside

. . . Or why would you listen to us?

Our goals in this talk

We will argue that

Our goals in this talk

We will argue that

1. Computers, considered as *automated formal systems*, suggest they have a unique ethical status.

Our goals in this talk

We will argue that

1. Computers, considered as *automated formal systems*, suggest they have a unique ethical status.
2. That there's an open philosophical problem in the applied ethics of formal methods (i.e., mathematically proving computers correct).

Our goals in this talk

We will argue that

1. Computers, considered as *automated formal systems*, suggest they have a unique ethical status.
2. That there's an open philosophical problem in the applied ethics of formal methods (i.e., mathematically proving computers correct).
3. Also, we will try to give you one practitioner's perspective on formal methods applications today.

What we do **NOT** want to convince you of

It is not our goal to

What we do **NOT** want to convince you of

It is not our goal to

1. Promote formal methods or argue that formal methods should replace other kinds of system validation (e.g., random testing, MC/DC coverage, etc.).

What we do **NOT** want to convince you of

It is not our goal to

1. Promote formal methods or argue that formal methods should replace other kinds of system validation (e.g., random testing, MC/DC coverage, etc.).
2. Proscribe a particular ethical theory of formal verification.

What we do **NOT** want to convince you of

It is not our goal to

1. Promote formal methods or argue that formal methods should replace other kinds of system validation (e.g., random testing, MC/DC coverage, etc.).
2. Proscribe a particular ethical theory of formal verification.
3. Retread debates over the "metaphysical status" of formal methods. (This was hashed out mostly in the late 80's by Fetzer & his commentators, Barwise, B.C. Smith, and others).

A warning to formal methods practitioners

Simplifying assumptions about are made throughout to extract the central philosophical issues.

What are formal methods?

A formal method is a tool or technique for formally proving (or disproving) a (mathematical model of a) computer implementation satisfies its specifications.

$$\frac{\varepsilon_3}{\varepsilon_1} = \frac{A'}{A^2}\beta^2$$

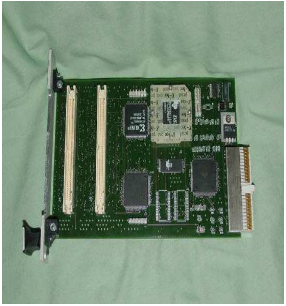
$$\varepsilon_1 = \left(\frac{A}{A+1}\right)^2 E_1$$

$$\mu_3 = \mu$$

$$\frac{\varepsilon_4}{\varepsilon_1} = \frac{A'}{A+1-A'}\frac{\varepsilon_3}{\varepsilon_1}$$

$$\mu_4 = \mu$$

$$\longleftrightarrow$$



(Intel's 1/2 billion reasons) why formal methods matter

Therac-25: A radiation-therapy machine killed or maimed 6 people in the 1980's due to software bugs.

(Intel's 1/2 billion reasons) why formal methods matter

Therac-25: A radiation-therapy machine killed or maimed 6 people in the 1980's due to software bugs.

Missle Defense: A 1960's early warning system falsely asserted that a full-scale nuclear attack by the Soviets had occurred due to unanticipated radiation from the moon.

(Intel's 1/2 billion reasons) why formal methods matter

Therac-25: A radiation-therapy machine killed or maimed 6 people in the 1980's due to software bugs.

Missle Defense: A 1960's early warning system falsely asserted that a full-scale nuclear attack by the Soviets had occurred due to unanticipated radiation from the moon.

Pentium FDIV Bug: It is estimated that a hardware bug in Intel's Pentium chip cost the company around 1/2 a billion dollars in the 1990's.

(Intel's 1/2 billion reasons) why formal methods matter

Therac-25: A radiation-therapy machine killed or maimed 6 people in the 1980's due to software bugs.

Missle Defense: A 1960's early warning system falsely asserted that a full-scale nuclear attack by the Soviets had occurred due to unanticipated radiation from the moon.

Pentium FDIV Bug: It is estimated that a hardware bug in Intel's Pentium chip cost the company around 1/2 a billion dollars in the 1990's.

Testing alone did not uncover these errors.
(Albeit we cannot claim that formal verification would have.)

Computers as automatic formal systems

Q: But why is mathematical proof so special for computers?

Computers as automatic formal systems

Q: But why is mathematical proof so special for computers?

A: *Automatic formal systems* (AFS) define a computer in terms of satisfying the following three properties [Haugeland 1989, Fodor 1990]:

Computers as automatic formal systems

Q: But why is mathematical proof so special for computers?

A: *Automatic formal systems* (AFS) define a computer in terms of satisfying the following three properties [Haugeland 1989, Fodor 1990]:

- Token manipulation: computers manipulate symbolic tokens according to formal rules (like games or logics).
- Digital: computers have exact, repeatable results, as opposed to continuous systems (e.g., billiards or the weather).
- Finite "playability": no computations take infinite time or require an oracle, etc.

Abstract vs. physical computers

In this talk, we are considering abstract computers.

- Abstract computers (are AFSes)
  - These are models that can be mathematically manipulated.
  - E.g., Turing Machines, Rewrite-formalisms, algorithms.
  - Realizable in a variety of mediums (e.g., silicon, Lincoln Logs, etc.).
  - But any realization should be behaviorally equivalent.
- Physical computers (that realize AFSes)
  - E.g., Digital wristwatches, laptops.
  - Can be pushed, prodded, and tested...
  - Only models of them can be mathematically manipulated.

Mind the (metaphysical) gap

- Abstract computers can be arbitrarily close to the physical computers (unlike, say, mathematical models of bridges or planes).
- The formal methods metaphysical debate principally centered around how small the gap is between abstract computers and concrete computers (for our purposes, we'll assume it's "sufficiently small").
- We call this assumption the Fundamental Formal Methods Hypothesis.

Mind the (metaphysical) gap (continued)

- Formally showing that a higher-fidelity model implements a more abstract one is called refinement.
- Digital systems allow for nearly arbitrary levels of refinement.
- The "many-models" paradox of AFSes: because the system can be modeled at so many levels of abstraction, ambiguity exists in the claim that a system is *formally verified*.

Q: If computers are AFSs, why not use formal methods all the time?

Q: If computers are AFSs, why not use formal methods all the time?

A: The model & proof of software is (very, very roughly) exponential in the conjunction of

- The size of the program.
- How "interesting" the properties to be proved are (e.g., divide by zero vs. termination).
- How "interesting" the program is—(real-time, concurrency, complicated semantics (e.g., object-oriented, complex types, etc.), exception-handling, runtime-systems, etc.).

Why not? Programs are huge

- In next-generation commercial aircraft (Airbus 380), there is an estimated one billion lines of code.
- A model with 10^{20} states is very small—this captures the behaviors of simple communication protocols. “Interesting” systems have an approximately-infinite state-space. (Today’s automated tools regularly handle state-spaces on the order of 10^{300}).

Why not? Digital systems are hard to verify

- Recall that a characteristic of AFSs is that they're digital.
- A difficulty of modeling large digital systems is that small changes to a program can mean big changes to the overall program properties:

  if a < b then ... vs.

  if a > b then ...

- This is the 2nd paradox of formal methods: digital systems are easy to model but hard to verify.

A note on digital systems (continued)

- Compare this to computational fluid dynamics: Small changes to an airfoil mean small changes to the aerodynamics.
- That is, models of continuous systems are usually compositional, whereas models of discrete systems are usually non-compositional.

Getting traction: economy vs. ethics

Economic—not ethical—motivations have driven large-scale formal methods adoption for the general consumer market. E.g.,

- Microsoft—maintaining market share by mitigating the perception of minimal security and numerous bugs.
- Intel, AMD, etc.: hardware can't be "patched" like software can, so mistakes are more costly.
- And others for "niche" uses: e.g., telecommunication protocols, language design, hardware compiler correctness, etc.

Safety-critical & security-critical software

Q: Why have the inroads been made there?

Safety-critical & security-critical software

Q: Why have the inroads been made there?

A:

- Mandated certification/evaluation: (e.g., DO-178B for FAA-certified software; Common Criteria for security-critical government systems).
- Economic motivation: à la the ultimate financial cost to Ford in the Pinto debacle.
- National security and military advantage.

But it's not clear to what extent ethical considerations are the driving force.

The “conventional” wisdom

Some formal methods practitioners have been waiting for the day they’d be heralded as prophets. Particularly in the 80’s, many believed that

- Lawsuits: software vendors would be held legally liable for faulty software (despite faulty software costing the U.S. economy some $5 billion annually.)
- Complexity: the complexity of systems could be managed only by formal proof.
  - Systems have too many states.
  - Safety-critical reliability requirements are too high (e.g., 10^{-9}hour for catastrophic error).
- Ubiquity: software system pervading medical devices, automobiles, aircraft, banks, etc. would necessitate higher assurance.

None became prime motivators. But, these issues may factor into a an ethical theory. . .

Traditional computer ethics

Our contention is that computer ethics research focuses on potentially novel aspects of physical computers, such as

- Persistent data storage.
- Rapid & widespread data transfer.
- Rapid and pervasive data analysis.
- The ubiquity of computers (e.g., nano-computers).

Other considerations for an "ethical theory of formal methods"

- Stallman's (et al.) call for open software.
- How culpability is divided amongst performers in software systems (e.g., architects, developers, formal methodists, integraters, managers, requirements developers, salespeople, testers, users, etc.). See Douglas Birsch, 2004.
- How formal methods is integrated with the overall validation of the system. Validation is about providing evidence that a system meets its specification. See John Rushby's 2007 articles on a *science of certification*.

Proposed outcomes

A significant contribution to computer ethics would be made by answering the following questions:

- (Historical/empirical) why has the "best engineering practice" of formal methods not become a part of software system development?
- What moral obligation is there to provide correctly functioning software and to provide evidence that this is so?
- Under what conditions should systems should be proved correct and what ethical obligations demand it?

Recent Related Work

Computers, justification, and mathematical knowledge

by Konstantine Arkoudas and Selmer Bringsjord. *Minds and Machines*, 2007.
Discusses philosophical issues of mechanical-proof certification.

Ethical protocols design

by Matteo Turilli. *Ethics and Information Tech.*, 2007.
Proposes a method for realizing ethical protocols.

Computer systems and responsibility: a normative look at technological complexity

by Debrah Johnson and Thomas Powers. *Ethics and Information Tech.*, 2005.
Investigates the special role of computer technology-assisted moral actions.

Moral responsibility for harm caused by computer system failures

by Douglas Birsch. *Ethics and Information Tech.*, 2004.
Investigates, by case-study of the Therac-25 incident, how and why humans are responsible in technology malfunctions.

Web Resources

Slides from this talk

`http://www.cs.indiana.edu/~lepike`

Google: lee pike

Online bibliography for the philosophical of formal methods

`http://www.cse.buffalo.edu/~rapaport/510/canprogsbeverified.html`

Google: rapaport programs verified