Proving Computer Systems Correct: (One Reason) Why Logic Matters

Supratik Chakraborty
IIT Bombay

Jan 8, 2025

Computers in Safety-Critical Systems

- Nuclear reactors, chemical plants, avionics, medical care, space missions, railway signaling, ...

Computers in Safety-Critical Systems

- Nuclear reactors, chemical plants, avionics, medical care, space missions, railway signaling, ...
- Enormous cost of failures induced by bugs
  - Loss of lives, environmental and infrastructure damage,
  - Financial loss, scientific setbacks, ...

Computers in Safety-Critical Systems

- Nuclear reactors, chemical plants, avionics, medical care, space missions, railway signaling, ...
- Enormous cost of failures induced by bugs
  - Loss of lives, environmental and infrastructure damage,
  - Financial loss, scientific setbacks, ...
- Corner-case bugs in software and hardware
  - Extremely hard to detect
  - May escape extended testing – infeasible to test all cases
  - Can cause catastrophic failure of safety-critical systems

Some Infamous Computer Bugs

- Therac-25 administers lethal radiation to patients (1985-87)
  - Bug in software controller
- Pentium FDIV bug costs Intel US$475 million (1994)
  - Bug in lookup-table of algorithm
- Ariane 5 rocket explodes a minute after launch (1996)
  - Software overflow bug
- NASA's Mars Climate Orbiter veers off and disintegrates (1999)
  - Software bug controlling spacecraft's motion
- Toyota Prius hybrid cars grind to halt unexpectedly (2015)
  - Software bug in car's embedded controllers
- Airbus A400M military transport aircraft crashes (2015)
  - Bug in software configuration
- Nest's IoT-enabled thermostat freezes homes in peak winter (2016)
  - Software bug in thermostat control

Role of Logic and Formal Methods

- Complexity of systems makes it humanly impossible to identify
  - All corner case inputs for software & hardware
  - Long sequences of inputs that can lead to failure

Role of Logic and Formal Methods

- Complexity of systems makes it humanly impossible to identify
  - All corner case inputs for software & hardware
  - Long sequences of inputs that can lead to failure
- Testing still pre-dominant method of validation in industry
  - *Strength:* Easier to generate test inputs, actual system can be tested
  - *Weakness:* Confidence only as good as test suite
  - E. W. Dijkstra: *"Program testing can be used to show the presence of bugs, but never their absence."*

Role of Logic and Formal Methods

- Complexity of systems makes it humanly impossible to identify
  - All corner case inputs for software & hardware
  - Long sequences of inputs that can lead to failure
- Testing still pre-dominant method of validation in industry
  - *Strength:* Easier to generate test inputs, actual system can be tested
  - *Weakness:* Confidence only as good as test suite
  - E. W. Dijkstra: *"Program testing can be used to show the presence of bugs, but never their absence."*
- Complementary role of logic based formal verification
  - *Strength:* Comprehensive, no test inputs, catches all bugs that exist
  - *Weakness:* Computationally harder, requires greater expertise, only models verified

Role of Logic and Formal Methods

- Complexity of systems makes it humanly impossible to identify
  - All corner case inputs for software & hardware
  - Long sequences of inputs that can lead to failure
- Testing still pre-dominant method of validation in industry
  - *Strength:* Easier to generate test inputs, actual system can be tested
  - *Weakness:* Confidence only as good as test suite
  - E. W. Dijkstra: *"Program testing can be used to show the presence of bugs, but never their absence."*
- Complementary role of logic based formal verification
  - *Strength:* Comprehensive, no test inputs, catches all bugs that exist
  - *Weakness:* Computationally harder, requires greater expertise, only models verified
  - Mandatory step before chip tape-out in Intel, AMD, Fujitsu, ...
  - Software companies rapidly adopting formal methods (Microsoft, IBM, TCS, ...)

Role of Logic and Formal Methods

- Complexity of systems makes it humanly impossible to identify
  - All corner case inputs for software & hardware
  - Long sequences of inputs that can lead to failure
- Testing still pre-dominant method of validation in industry
  - *Strength:* Easier to generate test inputs, actual system can be tested
  - *Weakness:* Confidence only as good as test suite
  - E. W. Dijkstra: *"Program testing can be used to show the presence of bugs, but never their absence."*
- Complementary role of logic based formal verification
  - *Strength:* Comprehensive, no test inputs, catches all bugs that exist
  - *Weakness:* Computationally harder, requires greater expertise, only models verified
  - Mandatory step before chip tape-out in Intel, AMD, Fujitsu, ...
  - Software companies rapidly adopting formal methods (Microsoft, IBM, TCS, ...)
  - Significant progress in last 30 years
    - Multiple Turing Awards for work related to formal verification
    - Formal verification tool-chains routinely used in several companies
    - Formal verification tools available commercially & in public-domain

Formal Verification in a Nutshell

- **System Modeling:** Given a system, derive a *mathematically precise* description of *what it does*
  - Automata, process calculi, programs, logics, circuits, ...
  - Automatically derived and/or manually constructed

Formal Verification in a Nutshell

- **System Modeling:** Given a system, derive a *mathematically precise* description of *what it does*
  - Automata, process calculi, programs, logics, circuits, ...
  - Automatically derived and/or manually constructed
- **Property Specification:** A *mathematically precise* description of *what the system is supposed to do*
  - Unambiguous objective interpretation
  - Automata, logics, constraint languages, structured English, ...
  - Free English can be ambiguous and not suitable

Formal Verification in a Nutshell

- **System Modeling:** Given a system, derive a *mathematically precise* description of *what it does*
  - Automata, process calculi, programs, logics, circuits, ...
  - Automatically derived and/or manually constructed
- **Property Specification:** A *mathematically precise* description of *what the system is supposed to do*
  - Unambiguous objective interpretation
  - Automata, logics, constraint languages, structured English, ...
  - Free English can be ambiguous and not suitable
- **Verification Techniques:** Computational techniques to check if *model* satisfies *specification*

Formal Verification in a Nutshell

- **System Modeling:** Given a system, derive a *mathematically precise* description of *what it does*
  - Automata, process calculi, programs, logics, circuits, ...
  - Automatically derived and/or manually constructed
- **Property Specification:** A *mathematically precise* description of *what the system is supposed to do*
  - Unambiguous objective interpretation
  - Automata, logics, constraint languages, structured English, ...
  - Free English can be ambiguous and not suitable
- **Verification Techniques:** Computational techniques to check if *model* satisfies *specification*
  - Curse of undecidability
    - General algorithms: **mathematical impossibility**
    - (Semi-)algorithms for practically relevant sub-classes of problems
    - Tools require significant engineering

Formal Verification in a Nutshell

- **System Modeling:** Given a system, derive a *mathematically precise* description of *what it does*
  - Automata, process calculi, programs, logics, circuits, ...
  - Automatically derived and/or manually constructed
- **Property Specification:** A *mathematically precise* description of *what the system is supposed to do*
  - Unambiguous objective interpretation
  - Automata, logics, constraint languages, structured English, ...
  - Free English can be ambiguous and not suitable
- **Verification Techniques:** Computational techniques to check if *model* satisfies *specification*
  - Curse of undecidability
    - General algorithms: **mathematical impossibility**
    - (Semi-)algorithms for practically relevant sub-classes of problems
    - Tools require significant engineering
  - Theorem proving
    - Axioms and inference rule schemas for specific logics
    - Apply (semi-)automatically to derive consequence from premises

Formal Verification in a Nutshell

- **System Modeling:** Given a system, derive a *mathematically precise* description of *what it does*
  - Automata, process calculi, programs, logics, circuits, ...
  - Automatically derived and/or manually constructed
- **Property Specification:** A *mathematically precise* description of *what the system is supposed to do*
  - Unambiguous objective interpretation
  - Automata, logics, constraint languages, structured English, ...
  - Free English can be ambiguous and not suitable
- **Verification Techniques:** Computational techniques to check if *model* satisfies *specification*
  - Curse of undecidability
    - General algorithms: **mathematical impossibility**
    - (Semi-)algorithms for practically relevant sub-classes of problems
    - Tools require significant engineering
  - Theorem proving
    - Axioms and inference rule schemas for specific logics
    - Apply (semi-)automatically to derive consequence from premises
  - Model checking
    - Algorithmically check if all runs of model satisfy specification
    - Admits automation for special classes of problems.

Formal Verification in a Nutshell

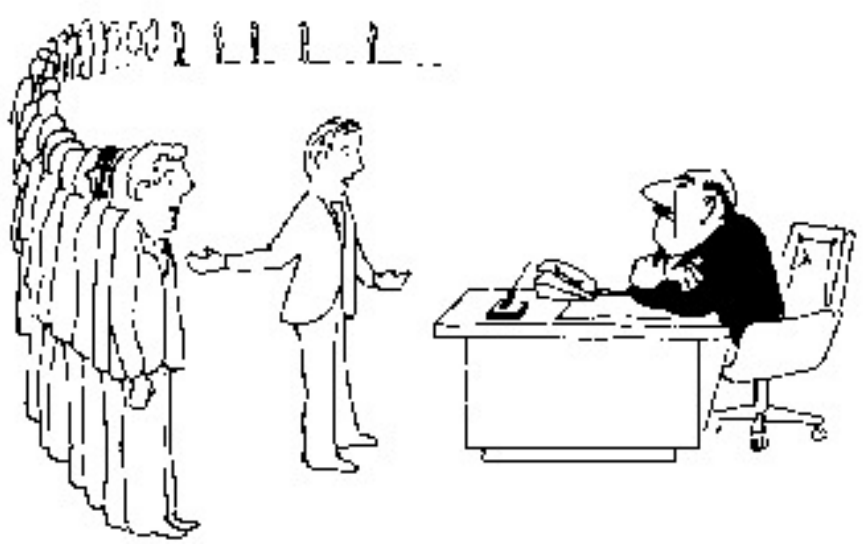
- **System Modeling:** Given a system, derive a *mathematically precise* description of *what it does*
  - Automata, process calculi, programs, logics, circuits, ...
  - Automatically derived and/or manually constructed
- **Property Specification:** A *mathematically precise* description of *what the system is supposed to do*
  - Unambiguous objective interpretation
  - Automata, logics, constraint languages, structured English, ...
  - Free English can be ambiguous and not suitable
- **Verification Techniques:** Computational techniques to check if *model* satisfies *specification*
  - Curse of undecidability
    - General algorithms: **mathematical impossibility**
    - (Semi-)algorithms for practically relevant sub-classes of problems
    - Tools require significant engineering
  - Theorem proving
    - Axioms and inference rule schemas for specific logics
    - Apply (semi-)automatically to derive consequence from premises
  - Model checking
    - Algorithmically check if all runs of model satisfy specification
    - Admits automation for special classes of problems.
  - Hybrid techniques: intelligent combinations of above

Everyday Reality of Formal Verification

I can't find an efficient algorithm, but neither can all these famous people.

[Courtesy: *Computers and Intractibility: A Guide to the Theory of NP-Completeness, M.R. Garey and D.S. Johnson, 1979*]

Everyday Reality of Formal Verification

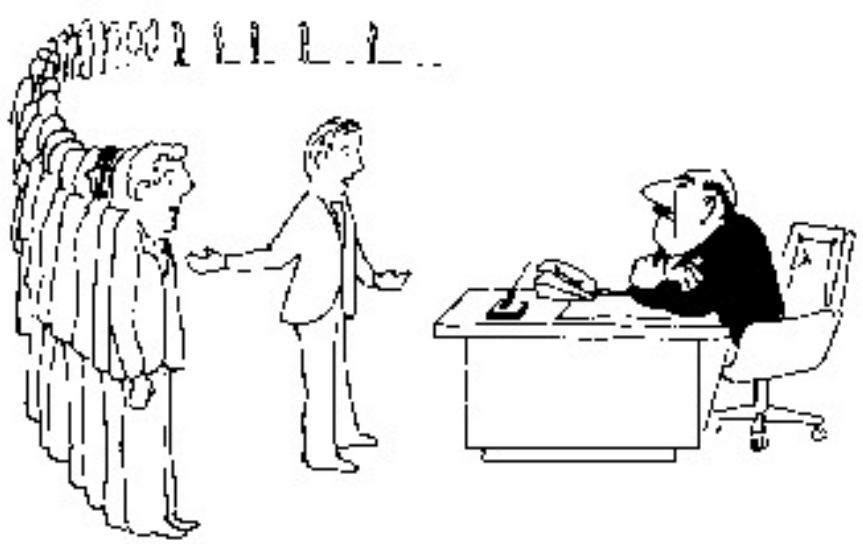


I can't find an efficient algorithm, but neither can all these famous people.

[Courtesy: *Computers and Intractibility: A Guide to the Theory of NP-Completeness, M.R. Garey and D.S. Johnson, 1979*]

Nevertheless, safety-critical systems must verifiably satisfy their specs

Everyday Reality of Formal Verification



I can't find an efficient algorithm, but neither can all these famous people.

[Courtesy: *Computers and Intractibility: A Guide to the Theory of NP-Completeness, M.R. Garey and D.S. Johnson, 1979*]

Nevertheless, safety-critical systems must verifiably satisfy their specs

- Developing logic-based reasoning techniques necessary
- Carving out relevant sub-problems crucial
- Domain knowledge invaluable
- Engineering optimizations essential