Formal Methods: Objectives, Techniques, & Perspectives

Gerald Luettgen

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia

http://www.icase.edu/~luettgen
Mathematics in Engineering
(example: computational fluid dynamics)

real-world problem
airflow over wing
abstraction, modeling
mathematical model
partial diff. equations

calculation
mathematical results
numerical approx.

real-world results
improved wing design
concretization, transformation

solving, high complexity
Mathematics in Computer Science

- Compiler construction: formal languages & automata theory
- Compiler optimization: data-flow analysis & abstract semantics
- Type theory: higher-order logic & deduction
- Performance modeling: numerical analysis & queuing theory
- Cryptography: algebraic number theory

Where is the mathematics in the development and engineering of digital systems?
Why Is Digital Systems Engineering Special?

- Classical engineering:
  - systems have continuous behavior \((\text{differentiable functions})\)
  - defects arise from physical wearout & bad design

- Digital systems engineering:
  - system have abrupt behavior \((\text{discrete functions, logical formulas})\)
  - all defects are design errors

---

Rules of thumb are not applicable in the digital world!
(“build bridge twice as strong as calculated”)
Correctness of Real-world Digital Systems

Assume:

• a system consisting of 500 basic components
• each basic component is 99% correct
• all basic components are independent (idealistic assumption)

Probability of an overall correct system:

• \(0.99^{500} = 0.66\%\)

→ Complex systems are incorrect!
Testing of Ultra-reliable Systems

To measure a $10^{-9}$ probability of failure for a 1 hour mission,
one must test for $10^9$ hours = 114,000 years

Testing is infeasible for ultra-reliable systems!

(In practice it is hard to justify testing beyond $10^{-4}$ or $10^{-5}$.)
Engineering Life Cycle & Quality Assurance

- requirements
- design
- implementation
- validation
- integration & maintenance

- review & prototyping
- review & inspection
- parsing & type checking
- testing & debugging
- documentation of every step
Experiences With Fault Rates
(based on JPL Data)

- Rule of thumb:
  - 1 major fault every 3 pages of requirements
  - 1 major fault every 21 pages of code

- Fault statistics for the Voyager and Galileo projects:
  - coding faults: 6% of overall faults (!!!)
  - function faults: 71% (most of them due to requirements problems)
  - interface faults: 23% (due to poor communication between teams)

- Most hazardous faults discovered during system integration!
Testing & Formal Methods

- RP: requirements phase
- DP: design phase
- IP: implementation phase
- VP: validation phase
- MP: integration & maintenance phase

Costs for eliminating errors decrease significantly after system delivery, especially when formal methods are applied. Effort increases as the system progresses through the phases, with formal methods and testing being key factors in managing this increase.
Outline of This Talk

- What are Formal Methods and how can they help?
- What are the employed techniques?
- What about technology transfer to industry?
- What are the mid-range perspectives?
Formal Methods in a Nutshell

- **Objectives:** help engineers construct more reliable systems
- **Field:** cuts across most areas in Computer Science
- **Foundations:** logic and discrete mathematics
- **Applications:** specification, design, & verification of hardware & software

→ A novel approach to quality assurance
Application Areas of Formal Methods

Safety- & life-critical systems:
- avionics e.g., flight guidance systems (Rockwell Collins)
- aerospace e.g., Space Shuttle software (NASA, Boeing)
- medicine e.g., heart pacemakers (Cardiac Pacemakers)
- automotive industry e.g., anti-lock brakes (Daimler-Chrysler)
- railway e.g., signaling systems (British Rail)

Money-critical systems:
- telecommunications e.g., phone switches (AT&T)
- computer chips e.g., processors (Intel, Cadence)
- e-commerce e.g., databases & security (IBM)
What Are Formal Methods Not?

- Guarantors for correct hardware and software
- Replacements for testing and reviews
- Just structured methods

Instead Formal Methods are:

- Mathematical techniques for reasoning about system behavior
  - especially for embedded and concurrent systems
- Advanced debugging aids
Formal Methods in the Past and Today

▶ 15 years ago: FM were expensive and purely academic
  • but no other ways to ensure correctness

critical systems have been produced without FM

▶ Today: It is practical to produce systems using FM
  • detect errors early
  • reduce expensive system re-design

→ Save money, time, & lives!
Formal Specification

1. A specification written and approved in accordance with established standards.

2. A specification written in a formal notation, often for use in proof of correctness.

(IEEE standard glossary of SWE terminology)

- Specification languages:
  - logical       e.g., PVS       (higher-order logic & funct. constructs)
  - declarative  e.g., Z          (set theory)
  - algebraic    e.g., LOTOS      (process algebra)
  - operational  e.g., Statecharts (state machines)
Theorem Proving

Specify:
- the system
- the required property
- the assumptions and background theories

Prove: $\text{|- (background & assumptions & system) } \Rightarrow \text{ property}$
by symbolic deduction (proof checking)

Experience:
- works best for data-intensive systems
- requires user interaction
- hard to read diagnostic information (unsuccessful proof sequents)
- use for verification

as formulas in a single logic
Model Checking

Specify:

- the system as a finite-state machine
- the required property & the assumptions as formulas in a temporal logic

Establish: \[ \text{system} \models (\text{assumptions} \implies \text{property}) \]

by an exhaustive search over the system’s state space

Experience:

- works best for control-intensive systems
- fully automatic, but system must be made finite-state
- easy to understand diagnostic information (traces leading to errors)
- use for debugging
Theorem Proving vs. Model Checking
(current research topics)

- Make both techniques work together:
  - Provide a finite-state abstraction of the considered system
  - Use theorem proving to show that abstraction preserves property of interest
  - Use model checking to verify the property for the abstraction

- Aggressive abstraction is often beneficial in early stages:
  - Generally find more errors by exploring all behaviors of a simplified model than some behaviors of more complex models
Obstacles for Technology Transfer

Common problems of novel techniques:

- Proper evaluation in industrial context is missing
- Lack of mathematical sophistication of (software) developers
- High initial costs, especially for consulting and training
- Unavailability of industrial-strength tools
- Domain-specific mathematics and methods are needed
Roles of Formal Methods

- **Analytical:**
  - whether a certain description is consistent
  - whether certain properties are consequences of proposed requirements
  - whether one level of design implements another

- **Descriptive:**
  - clarify document requirements or designs
  - facilitate communication during inspections and reviews

- **Mandatory:**
  - provide assurance or certification data
Formal Methods Light

- Analysis rather than verification:
  - partial analysis of partial specifications
  - no pursuit of perfection or completeness
  - debugging by posing challenges

- Separated from a project’s mainstream:
  - experienced consultants address Formal Methods issues

⇒ Essential for initiating technology transfer to industry
Example for Technology Transfer: Space Shuttle GPS Change Request

- GPS navigation added to the Shuttle (110 pages of change requests)
  - GPS data loaded as if state-vector updates from ground radar
- Change request focuses on interfaces to existing components
- Two principal functions formally specified as state machines in the language of the theorem prover \textit{PVS} (3,300 lines)
- Anomalies discovered through the process of formalization and automated theorem proving (sophisticated type checking)
Result of “Debugging” GPS Change Request

- Numbers of discovered anomalies part way through:
  - Formal Methods: 30 (7 major)
  - Human Review: 4 (1 major)

- Anomalies revealed by Formal Methods at completion:
  - High major: 1 (requirement cannot be implemented)
  - Low major: 10 (requirement does not reflect author’s intent)
  - High minor: 65 (requirements are confusing)
  - Low minor: 10 (minor documentation changes)

- Effort: 4 staff-months (including 2 months for preparation)
Another Example:
Space Shuttle 3E/O Change Request

- 3E/O (Three-Engines Out) is an abort maneuver invoked when all 3 main engines fail during ascent (70 pages of change requests)

- Change request focuses on
  - mode changes of the fault-handling logic
  - interaction of the logic with its highly nondeterministic environment

- Behavioral model specified in the state-machine language of the model checker Murphi (1200 lines)

- Anomalies discovered through the process of formalization and invariant verification
Result of Model Checking 3E/O Change Request

- Numbers and types of discovered anomalies:
  - undocumented assumption: 3
  - inconsistent terminology: 10
  - redundant calculation: 2
  - missing initialization: 1
  - interface anomaly: 2
  - logical error: 1

- Only the logical error was also found by human review!

- Effort: 3 1/2 staff-months (including 2 months for preparation)
Reported Benefits of Formal Methods

- Reveal faults in early stages of the life cycle
- Find subtle design errors that are otherwise overlooked
- Support unambiguous communication and documentation
- Fundament reviews (consensus) by analyses (calculation)
- Force attention to detail

Formal Methods are worth investigating
Formal Methods & Certification: DO-178B

DO-178B by RTCA (Requirements and Technical Concepts in Aviation):

- “FAA’s certification requirements” for avionics software
- Based on rigorous software development process, including documentation, review, & analyses of each life-cycle phase
- Allows FM as “alternative means of compliance”

Some excerpts from the document:

- “The goal of applying FM is to prevent and eliminate specification, design, and implementation errors throughout software development.”
- “FM are complementary to testing…”
- “Application of FM can start and stop with (any) consecutive levels in the design structure.”
Conclusions

- Use Formal Methods to find bugs in systems:
  - is computationally easier than proving correctness
  - scale systems down
  - employ powerful and highly automated tools

- Target Formal Methods to specific applications:
  - control-dominated systems are harder to prove than data-dominated
  - in hardware: cache coherence, pipelines, floating point arithmetic
  - in software: fault-tolerance & real-time issues of concurrent systems
Perspectives

What you should expect to see in the not too distant future:

- New-generation tools:
  - integrated with existing tools (e.g., Statemate® & model checkers)
  - more heterogeneous, domain-oriented

- Metrics for Formal Methods:
  - will elaborate with industrial experience

- Textbooks and other training material

- Integration with engineering standards:
  - included in certification standard for avionics hardware ("RTCA SC-180")
Literature

Thanks!