Formal Methods
Past, Present, and Future

Jim Woodcock FREng

University of York
Southwest University, Chongqing
Aarhus University

20 March 2019

Huawei Research Forum, Bilzen

Outline

Formal Methods: Definitions

Formal Methods: Past

Formal Methods: Present

Formal Methods: Future

"There's nothing so practical as a good theory."

Kurt Lewin, 1951

founder of social psychology

Outline

Formal Methods: Definitions

Formal Methods: Past

Formal Methods: Present

Formal Methods: Future

Formal Methods: Analogy with Engineering Mathematics

- engineers traditionally build mathematical models of designs
- calculation establishes
  - design + modelled environment satisfies requirements
- only really useful when mechanised (eg, CFD)
- used in the design loop (exploration, debugging)
  - model, calculate, interpret, repeat
- also used in certification for assurance
  - evidence that model satisfies certain requirements
- need to be sure
  - model faithfully represents design
  - design is implemented correctly
  - environment is modelled faithfully
  - calculations are performed without error

Formal Methods: Analogy with Engineering Mathematics

- formal methods: same idea, applied to computational systems
- applied maths of Computer Science: formal logic
- models are formal descriptions in some logical system
  - program: set of instructions to a machine
  - reinterpret as a mathematical formula
- calculation mechanised by automated deduction
  - theorem proving, model checking, static analysis, etc
- formal calculations (can) cover all modelled behaviours
- if the model is accurate, this provides verification
- if the model is approximate, this provides debugging

Formal Calculations

- experience in finding high-value bugs
  - approximate model vs more accurate model
- searching massive spaces of discrete possibilities
- use human guidance
  - interactive theorem proving: PVS, Coq, Isabelle
- restrict attention to specific kinds of problems
  - model checking focuses on state machines
- use approximate models, incomplete search
  - model checkers are often used this way
- aim at something other than verification
  - bug finding, test case generation
- verify weak properties
  - static analysis
- give up soundness and/or completeness
  - static analysis

Outline

Formal Methods: Definitions

Formal Methods: Past

Formal Methods: Present

Formal Methods: Future

Survey of Industrial Use of FMs

- Jim Woodcock, et al.: Formal methods: Practice and experience. ACM Comput. Surv. 41(4): 19:1-19:36 (2009)
- snapshot of the state of the art in 2009 since extended
- originally 62 projects, 20-year span, now $>$ 120 projects
- most comprehensive survey of industrial practice in FM ever
- increasing frequency of FM projects
- application domains:
  - nuclear, health care, consumer electronics, space, semantic web, resource planning, automated car parking, embedded software, manufacturing & engineering, telecoms
- 10 software development tools
- applications:
  - real-time, distributed & transaction processing, parallel programming
  - hardware, control engineering, HCI, service-oriented computing, graphics

Project Details: Standards & Sizes

- requiring certification standards (34%):
  - general: IEC 61508
  - security: Common Criteria, ITSEC E6
  - railways: CENELEC EN50128
  - avionics: DO-178B
  - defence: DS 00-55 & 00-56
  - floating point numbers: IEEE 754
- equal split: 1–10 KLOC, 10–100 KLOC, and 100–1000 KLOC
- verification: mostly model checking
  - increased from 14% (1990s) to 59% (2000s)

Statistics

- Questionnaire completed by 62 major projects

	Improved	Worsened	No data
Time	30%	10%	60%
Cost	4%	6%	90%
Quality	92%	0%	8%
	Agree	Disagree	
Successful	94%	6%	
Satisfaction	98%	2%	
Tools could cope	89%	11%	
Would you repeat?	75%	25%	

- Rockwell Collins: repeated use

Grand Challenge in Verified Software

- Verified Software Initiative
- GC6: Dependable Systems Evolution
- Verified Software Repository
  - Mondex smart card
  - Flash File Store
  - OS Kernels
  - FreeRTOS
  - Radio Spectrum Auctions
  - Cardiac Pacemaker
  - Hypervisor
- Tokeneer
  - Janet Barnes, David Cooper, Rod Chapman
  - ISSSE, CACM papers
  - experiments: Andrew Ireland, Woodcock & Aydal
- Tokeneer is representative of what could be achieved in 2010

Tokeneer Overview

- TIS: Tokeneer ID Station
- project carried out by Praxis and SPRE for NSA
- the problem
  - Common Criteria for IT Security Evaluation
  - industry believes Common Criteria too expensive
- objectives
  1. cost-effective high-quality, low-defect EAL5 systems
  2. provide evidence for EAL5–EAL7
  3. controlled measurement of productivity and defect rates
  4. entire project archive at `www.adacore.com/tokeneer`
- rare opportunity to measure costs

Common Criteria Security Evaluation

- ISO/IEC 15408 standard for computer security certification
- Evaluation Assurance Level
  - EAL7: formally verified design and tested
  - EAL6: semi-formally verified design and tested
  - EAL5: semi-formally designed and tested
  - EAL4: methodically designed, tested, and reviewed
  - EAL3: methodically tested and checked
  - EAL2: structurally tested
  - EAL1: functionally tested
- EAL4-ish evaluation (1998): US$1–2.5 million

The Praxis method

- SPARK toolset
- semi-formal approach
- successful commercial applications of formal methods
- engineering method
  - addresses both assurance and cost requirements
- repeated successes
  - costs lower than traditional manual OO methods

- Praxis offered warranties on software delivered to customers

- software will behave as specified
- free bug fixes for first five years of operation

What Tokeneer Does

- NSA research project
- vehicle for investigating biometrics for access control
- control physical access to secure enclave
- user workstations within enclave
- users have smart card security tokens to gain access to enclave and to workstations
- readers outside enclave for cards and biometric tests
- pass the test and door opens for entry
- authorisation information on card for workstation access
- describes exactly the sort of access allowed for this visit
- times of working, security clearance, roles

Tokeneer Statistics

- Size and productivity

260 man days, 3 people part-time, 9 months elapsed

	LOC	contracts	LOC/day coding	LOC/day entire project
core SPARK	9,939	16,564	203	38
support Ada95	3,697	2,240	182	88

- cost for SPARK code: US$40–100/line

Tokeneer Statistics

Project phase	Effort %	Effort Person-days
Project management	11	28.6
Requirements	10	26.0
System specification	12	31.2
Design Core functions	15	39.0
TIS Core code and proof	29	75.4
System test	4	10.4
Support software and integration	16	41.6
Acceptance	3	7.8
Total	100	260.0

Project History

- two defects found in Tokeneer
- one found by Chapman using formal analysis
- one since delivery, discovered by code inspection
  - Spinellis's blog `www.spinellis.gr/blog/20081018/`
- testing team discovered two in-scope failures
- missing items in user manual
- testing really nearer 25%
- Woodcock/Aydal: 12 anomalous scenarios

Exceeding EAL5

- task set by NSA: conform to EAL5
- actually exceeded EAL5 requirements in several areas
  configuration control, fault management, testing

- EAL6—EAL7
  - EAL5: main body of core development work
  - EAL6&7: development areas covering specification, design, implementation, correspondence

- Why?

- because it was cheaper

Outline

Formal Methods: Definitions

Formal Methods: Past

Formal Methods: Present

Formal Methods: Future

2. Formal Methods: Present

- what's now the state of the art?

- seL4.verified microkernel
- Facebook Infer
- Amazon Web Services
- comparing Amazon & Facebook
- Altran/Praxis

seL4.verified

- NICTA: (formerly) Australia's ICT Research CoE
- machine-checked verification of seL4 microkernel down to C
- microkernel based on successful L4
  - 8,700 lines of C + 600 lines of assembly code
- proof assumes correctness of
  - compiler, assembler, boot code, cache management, hardware
- suitable for real-life use
  - performance $\approx$ best-performing microkernels
- proof that implementation formally satisfies specification
  - termination, crash-freedom, execution safety
  - formal correctness proof of access control mechanism
  - global variables and side effects, kernel memory management, concurrency and non-determinism (yielding, interrupts, exceptions), i/o & device drivers
- 1.4 py/kloc ($3\frac{1}{2}\times$Tokeneer)
- methodology for rapid kernel design and implementation

Facebook

- Facebook needs high-quality software
- Facebook runs on 62 Mloc (Feb 2017, excl. backend code)
  - bugs make users lose confidence
  - Dec 2018 bug: exposed 6.8M users' unposted photos to apps
- but pace of change makes it difficult to avoid errors
  - move fast and break things
- formal verification detects errors statically, before release
- difficult to deploy formal methods in industrial environments
- highly challenging with fast release cycles
- integration with Facebook's software development process
- features not wholly designed at the outset
- proposed, implemented, changed based on user feedback
- requires compositional program analysis
- feedback given to developers during incremental development

Facebook Infer

- open-source static code analysis tool
- originally developed by Monoidics, acquired by Facebook
- support for Java, C, C++, Objective-C
- deployed at Facebook to analyse Android and iOS apps
  - Facebook, WhatsApp, Instagram, Facebook Messenger
- based on Separation Logic for reasoning about
  - programs that manipulate pointer data structures
  - transfer of ownership
  - virtual separation between concurrent modules
- avoidance of semantic frame axioms
- modular reasoning
- essential to success in Facebook

Separation logic

- based on separating conjunction $*$
- formulae interpreted over program-allocated heaps
- $A * B$ holds of a heaplet that is divided into A and B

$$x \mapsto y * y \mapsto x$$

- x points to y and separately y points to x
- describes precisely two allocated memory cells with no aliasing
- just like mutation to computer memory
- reasoning about program commands
- update $*$-conjuncts in-place
- mimicks the operational in-place update of RAM

Separation Logic

- a specification
  - $\{r \mapsto open\}\, close(r)\, \{r \mapsto closed\}$
- suppose we have two resources r_1 and r_2
- described by $r_1 \mapsto open * r_2 \mapsto open$
- we close the first of them
  - $\{r_1 \mapsto open * r_2 \mapsto open\}\, close(r_1)\, \{r_1 \mapsto closed * r_2 \mapsto open\}$
- mention only one piece of state: small specification
- use that specification to update a larger precondition in place
- frame rule of separation logic
  - $$\frac{\{pre\}\, prog\, \{post\}}{\{pre * frame\}\, prog\, \{post * frame\}}$$
- key to the principle of local reasoning in separation logic

Facebook Infer

- abductive reasoning
  - start with an observation, then find the simplest explanation
  - discover frames to make entailments valid
  - essence of local reasoning
- Infer performance
  - 2013: 100s of bugs/month identified fixed before release
  - 2015: over 1000 bugs/month
  - millions of calls to internal theorem prover
- other users: Spotify, Uber, Mozilla, Sky, and Marks & Spencer

Amazon

- Transport Layer Security (TLS) encryption protocol
  - privacy and authentication in email, VoIP, browsing, messaging, cloud connections
- s2n: open source TLS implementation
  - Amazon and Amazon Web Services (AWS) products
  - S3: Simple Storage Service: 1.1M requests/s, 1 trillion objects
- financial services, government, pharmaceutical on AWS
  1. customers need high assurance on security in AWS
  2. automatic, continuous formal verification for rapid, cost-efficient development by distributed developer team
- verification must give guarantees with low effort
- proof and infrastructure for s2n's implementations of HMAC
  - Hash-keyed Message Authentication Code
  - ensures records can't be altered over open network
- Amazon's process: deep design & code reviews, static code analysis, stress testing, fault-injection testing, etc
- subtle bugs in complex concurrent fault-tolerant systems

Results

System	Components	Lines	Bugs
S3	Fault-tolerant low-level network algorithm	804 PlusCal	2+
	Background redistrib of data	645 PlusCal	1+1
DynamoDB	Replication & group membership system	939 TLA+	3
EBS	Volume management	102 PlusCal	3
Lock mgr	Lock-free data structure	223 PlusCal	0
	Fault tolerant replication and reconfig	318 TLA+	1

s2n Proof

- proof targets existing implementation
- updated either automatically or with low effort
- proof connects with existing proofs of security properties
- proof deployed in s2n continuous integration environment
- distributed development team modifying code
- repeated correctness proofs for every modification
- HMAC specification static since 2002
- existing mechanised proof (Beringer): high-level HMAC specification establishes cryptographic security property
- lower-level artifacts change more quickly
- implementation API, memory management, performance optimisations
- automated proof linked to stable foundational security results

s2n Proof

- Amazon/Galois: strong security properties for key components
- s2n implements pseudorandom function
- Coq proof + Galois Cryptol tool
- very small codebase: $\approx$ 550 lines of C
- Galois SAW tool: connection low-level Cryptol and s2n C code
- automatically verified, replayed on code changes
- pre/post specs supplied manually
- no loop invariants: bounded loops
- Hoare logic: code change effects localised
- 2017: proof replayed 956 times
- 3 manual interventions

Comparing Facebook & Amazon

- striking differences

- Facebook emphasises scalability: millions of lines of code
- targets simply stated properties
  - run-time errors
  - small theorems about big code
- uses fully automatic compositional reasoning

- Amazon s2n proof: few hundreds of lines of code
- targets complicated specifications
  - security properties
  - big theorems about small code
- uses manual interface specification for compositionality

- similarity: both efforts have unchanging specifications

Altran/Praxis

- safety-critical system verification with SPARK
- Agile engineering practices
- no assumption that product specification rarely changes
- CDIS UK ATC software
  - provides information about flight trajectories
  - used to detect potential position conflicts
- integration server rebuilds entire proof overnight
- populates persistent cache
- accessible to all developers following morning
- developers work on isolated change
- reprove entire system in 15 minutes on desktop PC
- reprove single module in seconds

Outline

Formal Methods: Definitions

Formal Methods: Past

Formal Methods: Present

Formal Methods: Future

Formal Methods: Future

- FM Future: The Hardware Vision & the Software Vision
- FM Technology: Boolean Satisfiability (SAT)
- FM Application: Security in Autonomous Systems

FM Future: The Hardware Vision

1. Verification will use 80% FV + 20% DV
2. Processors/tools will check/prove any design
3. Formal will replace simulation for unit testing
4. FV will support plug-and-play HW components
5. No constraints written by a human will be needed
6. Formal will be the sign-off tool
7. Formal will scale to full subsystems of very large chips
8. Formal will be able to specify properties at higher levels of design/abstraction

Correctness by Construction

- safety/security-critical domain
- Procurers In buying your next critical software system (e.g. something like DO-178B level A or Common Criteria EAL5 or above), aim for:
  1. Fewer than 0.1 defects per kloc, 1 year after delivery.
  2. Pay US$100 per line.
  3. Get a warranty.
- Regulators Regulate! Create a social and legal environment in which quality becomes the norm rather than the exception.
- Developers Bid projects based on the quality and productivity figures above. Bid them cheaper than a traditional process.

- extend this methodology to general software development

FM Technology: Boolean Satisfiability (SAT)

- simplest problem in automated reasoning
- find an assignment to make an arbitrary Boolean formula true
- first problem shown to be NP-complete
  - unless P = NP, there is no efficient algorithm
- two undergraduates + Sharad Malik at Princeton produced
  - Chaff SAT solver for many kinds of very large SAT problems
- now, SAT solvers routinely solve multi-million-variables
- real-world applications don't exhibit worst-case performance
- SMT: Satisfiability Modulo Theories (eg, Microsoft's Z3)
  - binary variables replaced by predicates over richer variables
- future developments
  - performance, parallelism, lazy and eager solvers, algebraic and model-finding approaches, leveraging external tools & techniques, developing new theories, ...
  - additional capabilities, fast incremental solving, interoperability with other tools, new applications, new domain-specific theories, non-CS applications: biology, finance, etc

FM Application: Security in Autonomous Systems

- example future application area for formal methods
- hacked robots perceive and affect the physical world
- proof-of-concept hacks of robots already demonstrated
- robotics central to many economic plans: UK, EU, US, China
- essential to new business models (eg, Uber)
- industry growth introduces new cyber-security challenges
- UK NCSC secure-by-design robotics & autonomous systems
- cost-effective development of verifiably secure robots
- domain-specific languages of attacks and security properties
- libraries of threats and trusted architectures
- proof, simulation, testing for novel attacks and architectures
- security, reactive, timed, & probabilistic properties
- security built in, not an afterthought in the implementation