Finding Bugs and Providing Assurance with Formal Methods

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

Why Do Systems Fail?

- Vulnerabilities are seldom due to gaping holes in principal functions
  - Testing finds these
- Usually due to subtle faults and unanticipated interactions
  - Exposed only by rare or complex scenarios
  - Testing is not so effective at finding these
- We need better ways to find these kinds of vulnerabilities
- And, ultimately, ways to provide assurance that none remain
- Automated formal methods provide effective ways to do this

Where Are The Faults Located?

- Usually in the *hardest* and *most critical* elements of design
  - Redundancy management and fault tolerance (flight control, nuclear power)
  - Group membership, replica determinism, consistency (protocols, distributed systems)
  - Cache coherence, superscalar dispatch, out-of-order execution (hardware)

- That is, distributed, concurrent, real-time systems operating in the presence of faults

- Number of possible behaviors is vast

- Testing can only visit a tiny fraction of them

Finding Bugs With Formal Methods

- Model-checking and its variants allows *all* the behaviors of a finite-state *model* of the system to be examined by sophisticated brute-force search

- Provides counterexample when violations founds

- Can examine vast numbers of states: hundreds of millions by explicit search, 10^{100} (sometimes) with symbolic methods

- Even so, have to "downscale" the system to obtain a model for which this is feasible (e.g., for a data transfer protocol, restrict queue size to 2, number of different message values to 3)

- Evidence is that examining *all* the behaviors of downscaled model is more effective at finding bugs that testing *some* of the behaviors of the real thing

Example (in SMV)

```
MODULE main
VAR
  semaphore : boolean;
  proc1 : process user(semaphore);
  proc2 : process user(semaphore);
ASSIGN
  init(semaphore) := 0;

MODULE user(semaphore)
VAR
  state : {idle,entering,critical,exiting};
ASSIGN
  init(state) := idle;
  next(state) :=      case
                        state = idle : {idle,entering};
                        state = entering & !semaphore : critical;
                        state = critical : {critical,exiting};
                        state = exiting : idle;
                        1 : state;
                      esac;
  next(semaphore) :=  case
                        state = entering : 1;
                        state = exiting : 0;
                        1 : semaphore;
                      esac;
FAIRNESS running
```

Example (continued)

```
Specification
  AG (proc1.state = entering -> AF proc1.state = critical)
is false, as demonstrated by the following execution sequence

state 1.1: semaphore = 0, proc1.state = idle, proc2.state = idle

state 1.2: [executing process proc1]
-- loop starts here --
state 1.3: proc1.state = entering

state 1.4: [executing process proc2]

state 1.5: [executing process proc2], proc2.state = entering

state 1.6: [executing process proc1], semaphore = 1, proc2.state = critical

state 1.7: [executing process proc2]

state 1.8: [executing process proc2], proc2.state = exiting

state 1.9: semaphore = 0, proc2.state = idle

resources used: user time: 0.1 s, system time: 0.1 s
BDD nodes allocated: 1134. Bytes allocated: 917504
```

Examples Where Bugs Have Been Found by Model Checking

- Futurebus+ cache coherence protocol (Clarke et al using SMV)
- SCI (Stern and Dill using Murϕ)
- Pentium FDIV, after the fact (Bryant)
- AT&T switch-to-switch protocol (Kurshan using Cospan)
- ACCESS.bus protocol (Boigelot and Godefroid using SPIN)
- Livelock in GNU UUCP
  (Smolka et al using the Concurrency Factory)
- MS Word paragraph formatting (Jackson using Nitpick)
- Space Shuttle requirements
  (Crow and DiVito using PVS and Murϕ)

Providing Assurance with Formal Methods

- Model checking is not so useful for verification as for refutation
  - "Downscaling" may be too aggressive

- Classical theorem-proving approaches to verification have recently become much more effective than previously
  - Effective combinations of decision procedures with interactive guidance
  - Sheer speed and horsepower
  - Focus on algorithmic rather than implementation issues

Examples Where Assurance Has Been Provided by Theorem Proving

- Part of ACCESS.bus protocol, after fixing bugs (Hooman using PVS)

- FLASH cache coherence protocol (Park using PVS)

- General theory of SRT division and specific radix 4 circuit and lookup table (Ruess using PVS)

- AAMP5 and AAMP-FV avionics processors (Srivas and Collins Commercial Avionics using PVS)

- Avionics fault tolerance (SRI, NASA, Allied-Signal, Honeywell, using PVS)

The Future

- Expect massive increases in performance and effectiveness
- Model checking
  - Symmetry and partial order reductions
- Theorem proving
  - More decision procedures (e.g., bit vectors)
  - Faster, more scalable decision procedures
  - Automated construction of invariants
- And combined methods
  - Integrated theorem proving and model checking
  - Automated abstraction to finite state

Looking Forward

- Automated formal methods will become the *most efficient and practical* ways to find bugs and to provide assurance for many kinds of systems

- The hardware and CAD companies are already actively exploring these techniques

- Software may not be so far behind

To Learn More

- Browse papers and technical reports at `http://www.csl.sri.com/fm.html`
  - `tse95.html` is a good overview: "Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS" from IEEE TSE February 1995
  - `chdl95-ms.html` is a paper on AAMP5 verification that was voted "best paper" at CHDL (the main hardware verification conference)
  - `cav95.html` describes the model checker integration in PVS
  - Links to Stanford papers and to Murϕ there also
- You can get PVS by anonymous FTP from `ftp.csl.sri.com` in `/pub/pvs/` (also from WWW link at URL above)
  - Allegro Lisp for SunOS 4.1.X, Solaris 5.X, IBM AIX 3.2.5
  - Need 24M memory, 60M swap space, Sparc 2 or better