SEHAS, Portland OR May 9 and 10, 2003.

From Reviews to Analysis: Challenge and Opportunity Converge

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, CA

New Challenges in Safety-Critical Applications

- Integrated modular avionics (IMA) and similar developments in other industries
- Previously, systems were federated
  - Meaning each function had its own computer system
  - Few connections between them

  So there were strong barriers to fault propagation
- Now, systems share resources
  - Processors, communications buses

  So need highly assured partitioning to restore barriers to fault propagation
- And they interact more intimately
  - E.g., braking, suspension, steering, on cars

  Raising concern about unintended emergent behavior

New Challenges in Regulatory Frameworks

- Integrated modular avionics
  - RTCA SC-200 and Eurocae WG60
- Want modular certification of separately qualified components
- It's not enough to show the components are "good"
  - Like the inertial measurement units of Ariane 4 and 5
- Need to be able to show the combination of components will be "good"
  - Unlike in Ariane 5
- This is what computer scientists call compositional reasoning
  - Deducing properties of the combination
  - From those of the components
  - Plus some "algebra of combination"

But compositional certification is different from compositional verification

New Challenges in Commercial Environments

- Need to reduce costs
  - Certification costs are about half of total
- And time to market
- Need to be able to upgrade and enhance already certified systems
- And want to be able to customize certified systems

Responding To The Challenges...

- Traditional methods for development, assurance, and certification of safety-critical systems are at their limits
- We need new methods for assurance and certification that are more efficient and more reliable
  - Move from reliance on process to evaluation of the product
- New methods should be less labor-intensive
  - Move from reviews
    - ⋆ Processes that depend on human judgment and consensus
  - To analysis
    - ⋆ Processes that can be repeated and checked by others, and potentially so by machine

This language is from DO-178B/ED-12B

So How Do We Analyze Software?

- Formal methods are about calculating properties of computer system designs
- Just like engineers in traditional disciplines use calculation to examine their designs
  - E.g., PDEs for aerodynamics, finite elements for structures
- So, with suitable design descriptions, we could use formal calculations to
  - Determine whether all reachable states satisfy some property
  - Determine whether a certain state is always achievable
  - Generate a (near) complete set of test cases

But Hasn’t That Been Tried and Failed?

Yes, it failed for three reasons

- No suitable design descriptions
  - Code is formal, but too big, and too late
  - Requirements and specifications were informal
  - Engineers rejected formal specification languages (e.g., ours)
- Narrow notion of formal verification
  - Didn’t contribute to traditional processes (e.g., testing)
  - Didn’t reduce costs or time (e.g., by early fault detection)
  - It was “all or nothing”
- Lack of automation
  - Couldn’t mechanize the huge search effectively
  - So needed human guidance—and interactive theorem proving is an arcane skill

But now there’s an opportunity to fix all that

The Opportunity

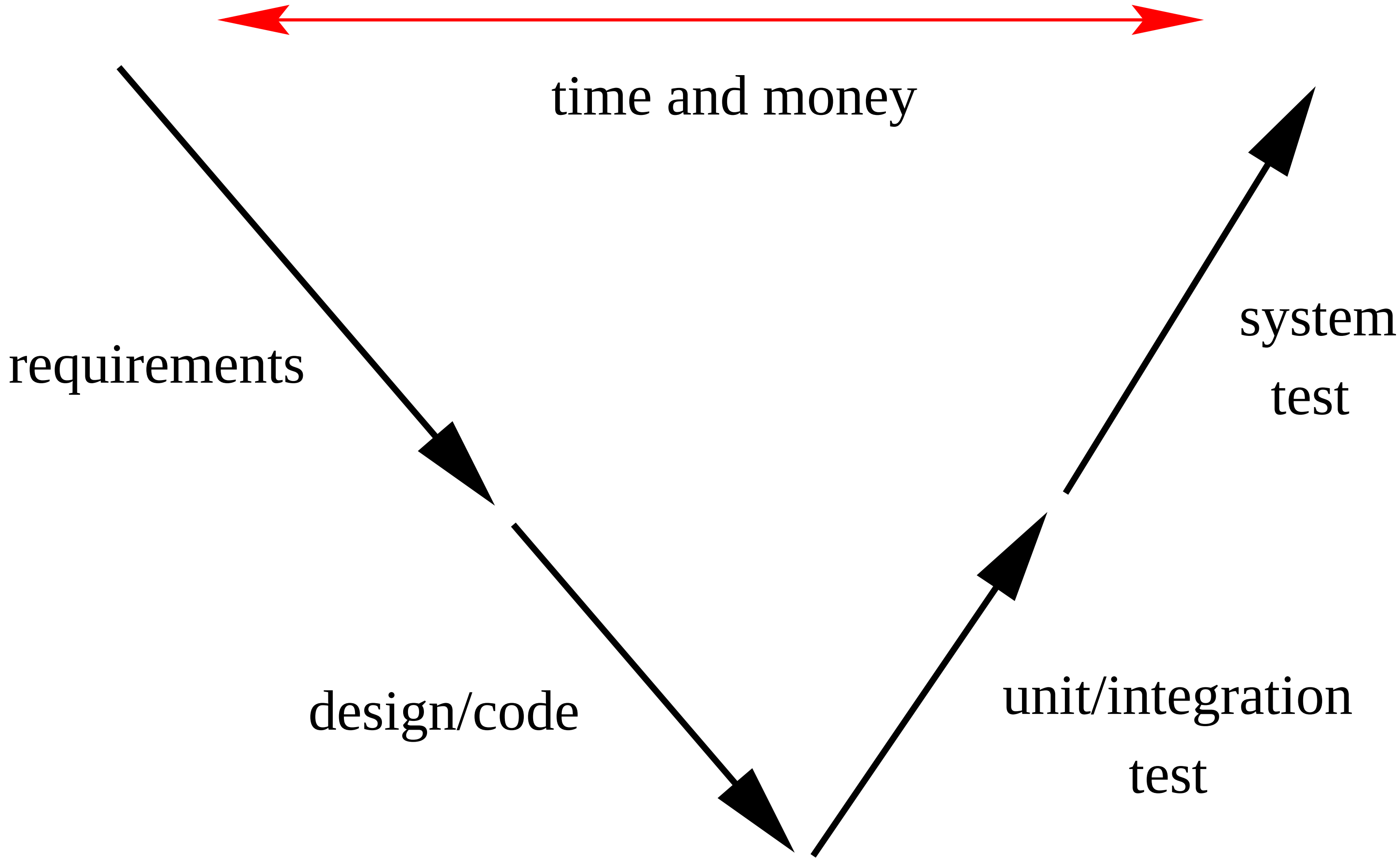
A convergence of three trends

- Industrial adoption of model-based development environments
  - Use a model of the system (and its environment) as the focus for all design and development activities
  - E.g., Simulink/Stateflow, SCADE and Esterel, UML
  - Some of these are ideal for formal methods (others are not, but can make do)
- New kinds of formal activities
  - Fault tree analysis, test case generation, extended static checking (ESC), runtime verification, environment synthesis, formal exploration
- More powerful, more automated deductive techniques
  - Approaches based on “little engines of proof”
  - New engines: commodity SAT, Multi-Shostak, “lemmas on demand”
  - New techniques: bounded model checking (BMC), k-induction, abstraction

New Kinds of Formal Analyses and Activities

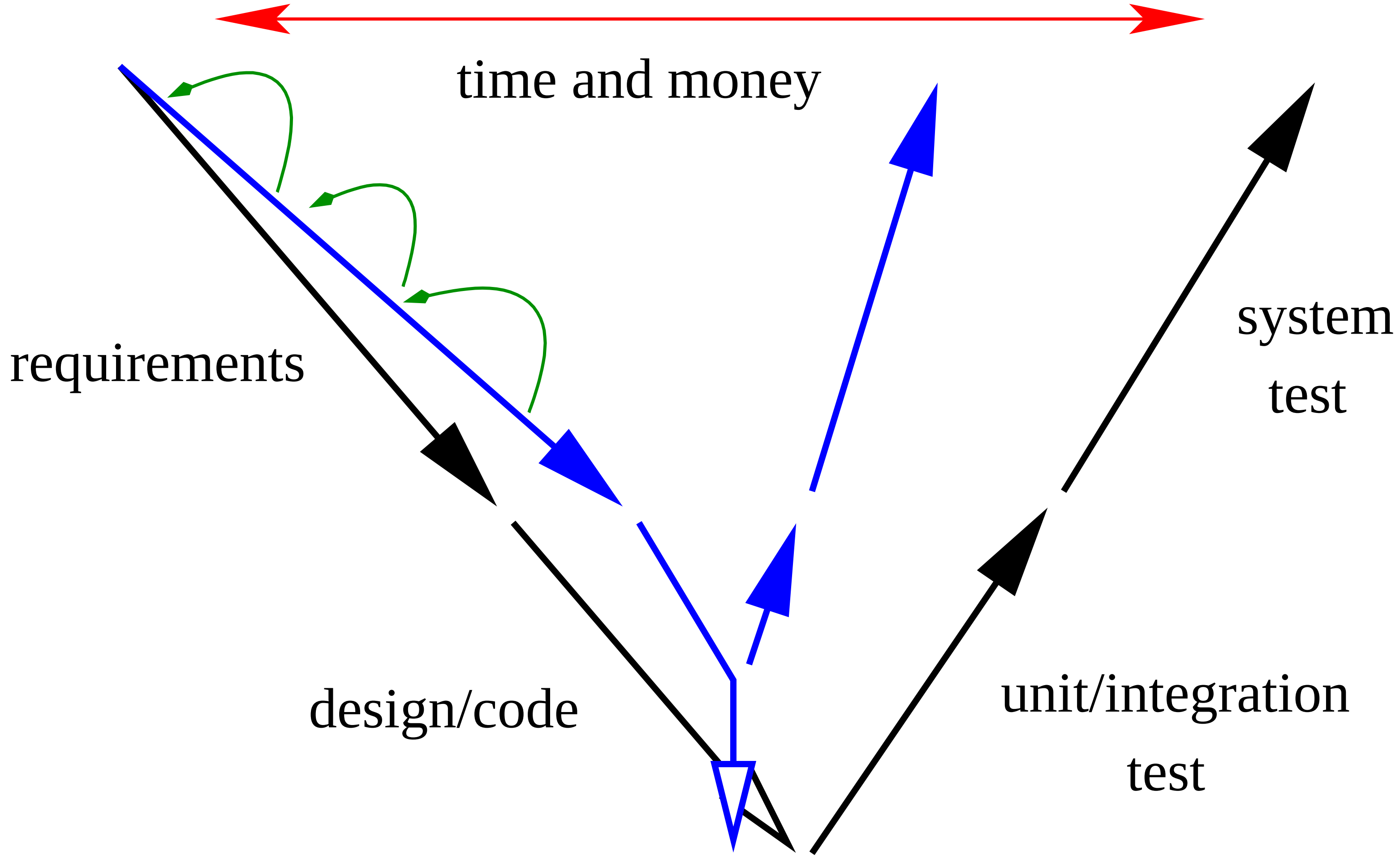
- Support design exploration in the early lifecycle
  - "Can this state and that both be active simultaneously?"
  - "Show me an input sequence that can get me to here with $x > y$"
- Provide feedback and assurance in the early lifecycle
  - Extended static checking, reachability analysis (for hybrid and infinite-state as well as discrete systems)
- Automate costly and error-prone manual processes
  - E.g., test case generation
- Together, these can provide a radical improvement in the traditional "V," in addition to that already provided by SCADE

Simplified Vee Diagram



Automated formal analysis can tighten the vee

Tightened Vee Diagram



Bounded Model Checking

- A key technology that finds many applications in tightening the Vee is bounded model checking (BMC)
- Is there a counterexample to this property of length k?
- Same method generates structural testcases
  - Counterexample to “there’s no execution that takes this path”

  And can be used for exploration
- Try $k = 1, 2, \ldots 100 \ldots$ until you find a bug or run out of resources or patience

Bounded Model Checking (ctd.)

- Given a system specified by initiality predicate I and transition relation T on states S, there is a counterexample of length k to invariant P if there is a sequence of states $s_0, \ldots, s_k$ such that

$$I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg P(s_k)$$

- Given a Boolean encoding of I and T (i.e., a circuit), this is a propositional satisfiability (SAT) problem
- Needs less tinkering than BDD-based symbolic model checking, and can handle bigger systems and find deeper bugs
- Now widely used in hardware verification

Infinite BMC

- Suppose T is not a circuit, but software, or a high-level specification
- It'll be defined over reals, integers, arrays, datatypes, with function symbols, constants, equalities, inequalities etc.
- So we need to solve the BMC satisfiability problem

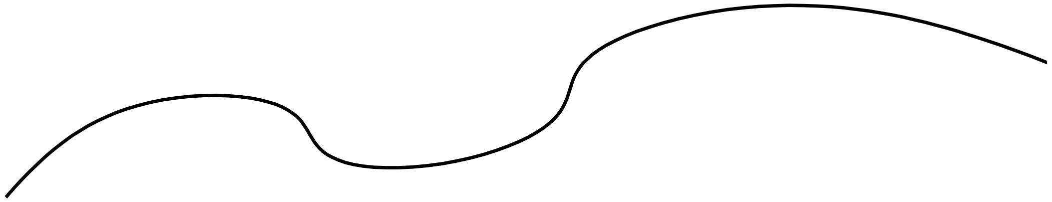
$$I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg P(s_k)$$

  over these theories
- Typical example
  - T has 1,770 variables, formula is 4,000 lines of text
  - Want to do BMC to depth 40
- These problems can be solved by combining decision procedures and a SAT solver

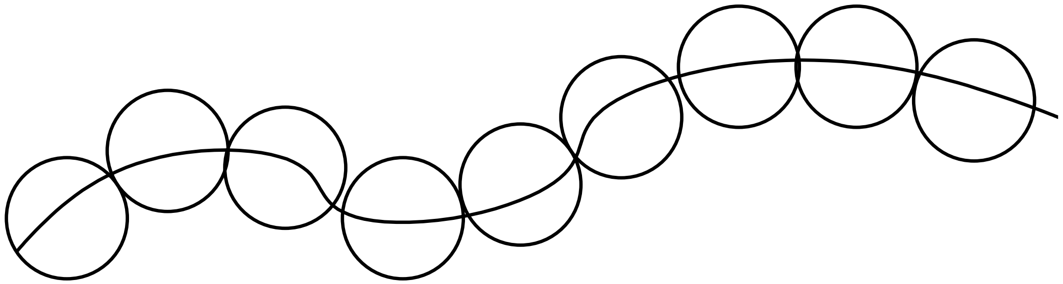
BMC Integrates With Informal Methods

- With big problems, may be unable to take k far enough to be interesting
- So, instead, start from states found during random simulation
- Can be seen as a way to amplify the power of simulation
- Or to extend its reach

Amplifying The Power Of Simulation



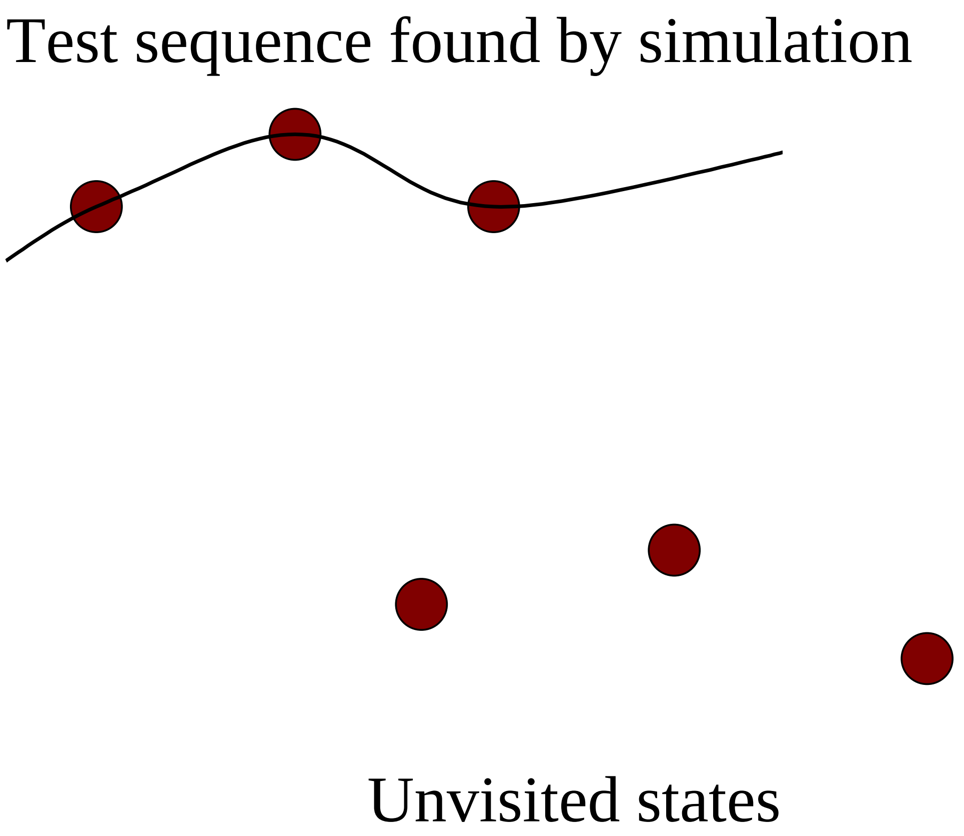
Test sequence found by simulation



Test sequence amplified by bounded model checking

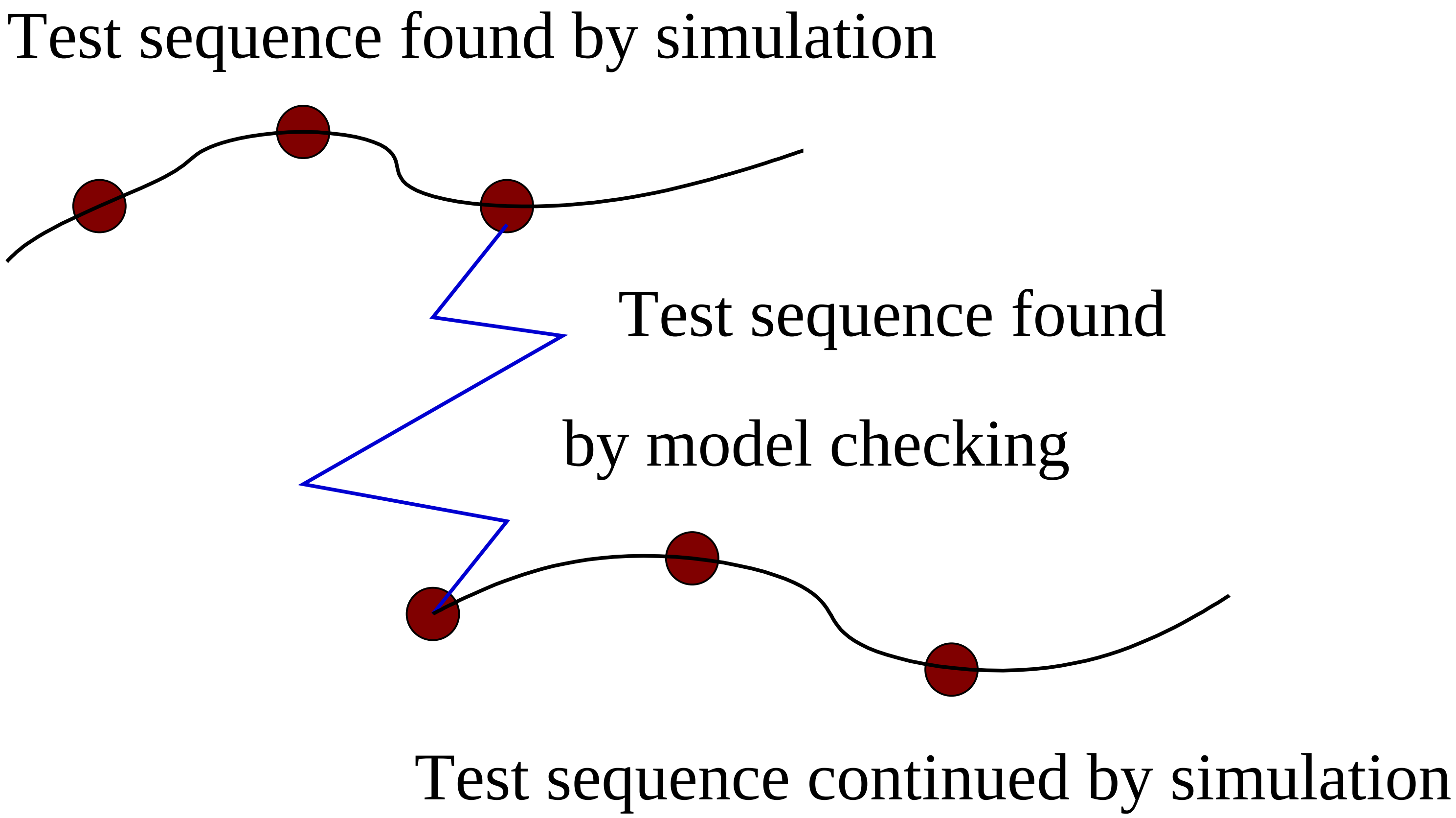
Extending The Reach Of Simulation

Random simulation can have trouble reaching some parts of the state space



Extending The Reach Of Simulation

So use BMC to jumpstart entry into those parts



Summary: Opportunity

- Model-based design methods are a (once-in-a-lifetime?) opportunity to get at artifacts early enough in the lifecycle to apply useful analysis within the design loop
- And formal analysis tools are now powerful enough to do useful things without interactive guidance
- The challenge is to find good ways to put these two together
  - Deliver analyses of interest and value to the developers
  - Or certifiers

  Can shift from technology push to pull

Summary: Technology

- The technology of automated deduction (and the speed of commodity workstations) has reached a point where we can solve problems of real interest and value to developers of embedded systems
- Embodied in our systems

  PVS.csl.cri.com: comprehensive interactive theorem prover

  ICS.csl.sri.com: embedded decision procedures

  SAL.csl.sri.com: (bounded) model checking toolkit
- And in numerous papers accessible from
  `http://www.csl.sri.com/programs/formalmethods/`