NICTA workshop, 29-31 May 2003 Sydney Australia, based on SEHAS, Portland OR 9 and 10 May 2003, and SCADE, Toulouse 19 May 2003.

Challenge and Opportunity in Mechanized Formal Methods

John Rushby

Computer Science Laboratory

SRI International

Menlo Park, California, USA

We Are Threatened By A Great Opportunity

- Industry is building more challenging applications
  - I'll focus on embedded systems
- But they are also changing the way they build them
- Creating an opportunity to insert formal methods
- Where by formal methods I mean calculating properties of formal descriptions of computer systems
  - I.e., the school represented at CAV, TACAS, SAS
  - Rather than FME, FSE, ICFEM, Z-B
- Later. . . new kinds of applications for formal methods

More and More Embedded Applications. . . And More Critical Ones

- More complete automation in mass transit
  - E.g., driverless trains
- More functions automated in airplanes
  - E.g., doors, escape slides
- More kinds of automation in airplanes
  - E.g., general aviation
- New industries automating critical functions
  - E.g., brake-, steer-by-wire in cars
- But the pool of talent and experience is small

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 compositional reasoning
  - Deducing properties of the combination
  - From those of the components
  - Plus some "algebra of combination"

But compositional certification differs from compositional verification

- Have to consider the plants and their hazards

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 analyses
        - ⋆ 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 fit the flow
  - 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), formal exploration, runtime verification, environment synthesis, controller synthesis
- 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

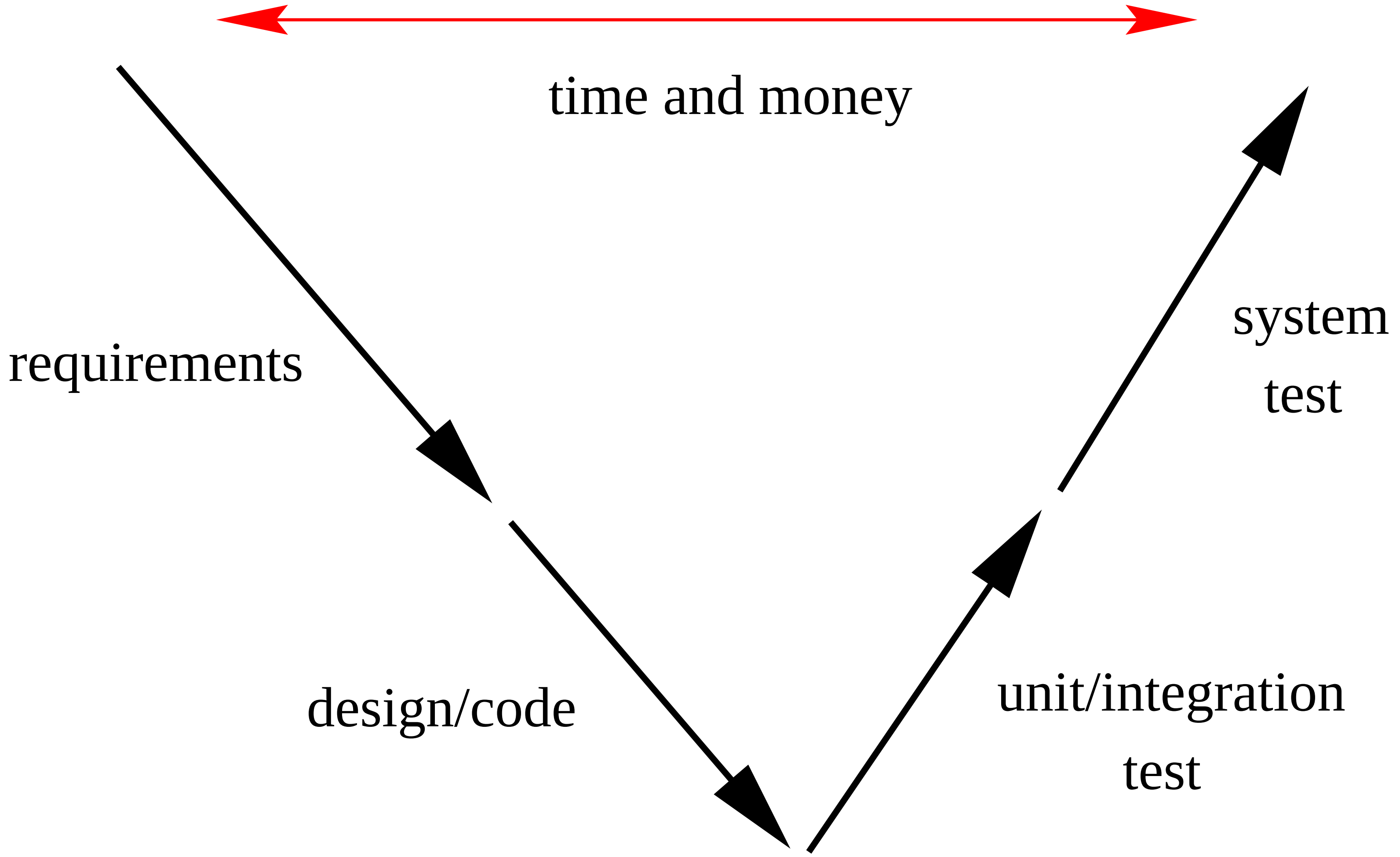
Industrial Adoption of Model-Based Development Environments

- These give access to formal descriptions throughout the lifecycle
- Being adopted at a surprisingly rapid pace
- A380 (SCADE), 7E7 (TBD) software will be developed this way
- 550,000 Matlab licences; how many UML?
- It was Ford that induced Mathworks to develop Stateflow
  - Has a ghastly semantics, but we have an adequate formalization
- Not just embedded systems
  - “Business logic”
  - System C and System Verilog: projections of 50,000 block designers, and 500,000 who assemble blocks
- Now, we just need to add analysis

New Kinds of Formal Analyses and Activities

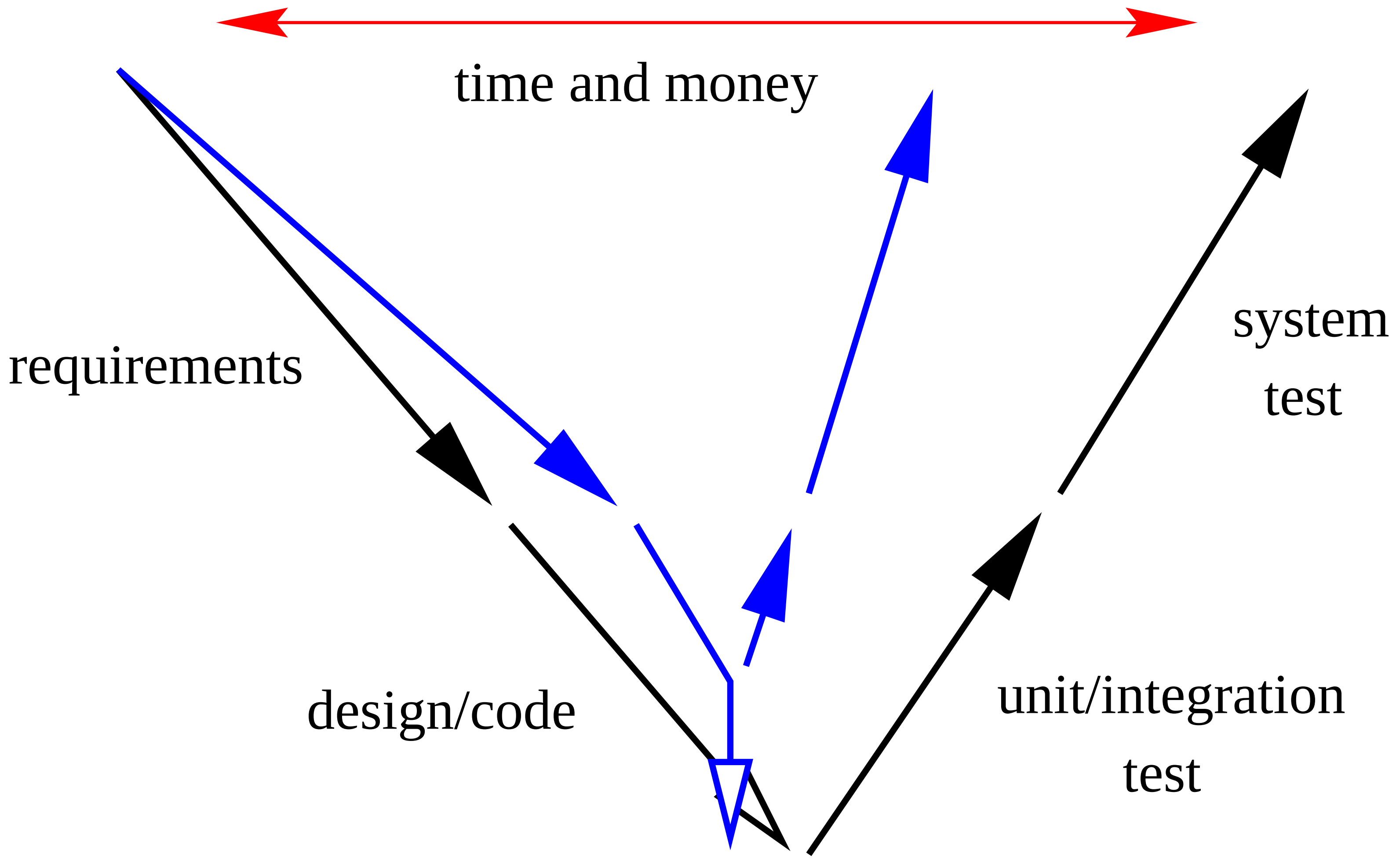
- Support design exploration in the early lifecycle
  - “Can this state and that both be active simultaneously?”
  - “Show me an input sequence that can get me to here with $x > y$”
- Provide feedback and assurance in the early lifecycle
  - Extended static checking
    - ⋆ Spark Examiner: 15,000 VCs, each may have 15,000 premises
  - 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”

Simplified Vee Diagram



Automated formal analysis can tighten the vee

Tighted Vee Diagram



Little Engines of Proof

- In the early lifecycle we have continuous quantities (real numbers and their derivatives), integers, other infinite and rich domains
- Later in the lifecycle, we have bounded integers, bitvectors, abstract data types
- Several of these theories are decidable, such as
    - Real closed fields
    - Integer linear arithmetic
    - Equality with uninterpreted functions
    - Fixed-width bitvectors

  The challenge is to decide their combination and to do it efficiently
- Need to make some compromises
    - The combination of quantified integer linear arithmetic with equality over uninterpreted functions is undecidable

  But the ground (unquantified) combination is decidable
- Combination methods were pioneered at SRI and Stanford more than 20 years ago, and we've continued to work on them ever since

Decision Procedures

- Tell whether a logical formula is inconsistent, satisfiable, or valid
- Or whether one formula is a consequence of others
  - E.g., does $4 \times x = 2$ follow from $x \leq y, x \leq 1 - y$, and $2 \times x \geq 1$ when the variables range over the reals?

  Can use heuristics for speed, but always terminate and give the correct answer
- Most interesting formulas involve several theories
  - E.g., does

  $$f(cons(4 \times car(x) - 2 \times f(cdr(x)), y)) = f(cons(6 \times cdr(x), y))$$

  follow from $2 \times car(x) - 3 \times cdr(x) = f(cdr(x))$?

  Requires the theories of uninterpreted functions, linear arithmetic, and lists simultaneously
- We want methods for deciding combinations of theories that are modular (combine individual decision procedures), integrated (share state for efficiency), and sound

Deciding Combinations Of Theories

- Our method (Shostak) works for theories that are canonizable and solvable
  - Almost any theory of practical concern
  - Others can be integrated using the slower method of Nelson-Oppen
- Yields a modular, integrated, sound decision procedure for the combined theories
  - First correct treatment published in 2002
  - Correctness has been formally verified in PVS (by Jonathan Ford)
  - Previous treatments were incomplete, nonterminating, and didn't work properly for more than two theories
  - ... Patent pending
- And the combination of canonizers is a canonizer for the combination
  - Independently useful—e.g., for compiler optimizations
  - Assert path predicates leading to two expressions; the expressions are common if they canonize to identical forms

Deciding Combinations Of Theories Including Propositional Calculus

- Capabilities just described tell whether one formula follows from several others
- Essentially, it's solving satisfiability for a conjunction of literals
- What if we have richer propositional structure
  - E.g., $x < y \wedge (f(x) = y \vee 2 * g(y) < \epsilon) \vee \ldots$ for thousands of terms
- We should exploit the efficient search strategies of modern SAT solvers
- So replace the terms by propositional variables
- Get a solution from a SAT solver (if none, we are done)
- Restore the interpretation of variables and send the conjunction to the core decision procedure
- If satisfiable, we are done
- If not, ask SAT solver for a new assignment—but isn't that expensive?

Deciding Combinations Of Theories Including Propositional Calculus (ctd.)

- Yes, so first, do a little bit of work to find some unsatisfiable fragments and send these back to the SAT solver as additional constraints (lemmas)
- Iterate to termination
- We call this "lemmas on demand" or "lazy theorem proving"
- Example, given integer x: $(x < 3 \wedge 2x \geq 5) \vee x = 4$
  - Becomes $(p \wedge q) \vee r$
  - SAT solver suggests $p = T, q = T, r = ?$
  - Ask decision procedure about $x < 3 \wedge 2x \geq 5$, it says No!
  - Add lemma $\neg(p \wedge q)$ to SAT problem
  - SAT solver then suggests $r = T$
  - Interpret as $x = 4$ and we are done
- It works really well
- But SAT solver must be specially engineered for this application
  - Gain orders of magnitude over loose combination with commodity SAT solver

ICS: Integrated Canonizer/Solver

- ICS is our implementation of everything just described
    - And a lot of things not described: proof objects, rich API

  ICS decides the combination of unquantified integer and real linear arithmetic, bitvectors, equality with uninterpreted functions, arrays, tuples, coproducts, recursive datatypes (e.g., lists and trees), and propositional calculus
- Core decision procedures are implemented in Objective Caml, SAT solver in C++
- The full system functions as a C library and can be called from virtually any language
- We have experience using it from C, C++, Lisp, Scheme, and Objective Caml
- Also has an interactive text-based front end
- Developed under Linux but ported to MAC OS X and to Windows XP (under cygwin)
- Freely available for noncommercial purposes under license to SRI
- Visit **`ics.csl.sri.com`** or **`ICanSolve.com`**

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
  - Though they generally use several methods in cascade

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
- Hey! That's exactly what ICS does
- . . . Patent pending

Infinite and Finite BMC

- Later lifecycle products replace infinite integers by fixed width bitvectors, etc.
- Can encode some of these datatypes in pure SAT
  - E.g., bitvectors as array of booleans, bounded integers as bitvectors
- Then provide SAT-level implementations of operations on them
  - E.g., hardware-like adders, shifters
- And that will semi-decide some combination of theories
- Exponentially less efficient than ICS decision procedures on many things where it does work (e.g., barrel shifter)
- But exact tradeoffs are fuzzy at lowest levels, and some applications will already split things up (e.g., arrays) before they send them to ICS
- So we're providing a “dial” that determines how much of the analysis for finite types is handled by decision procedures and how much by SAT

Extending (Infinite and Finite) BMC to Verification

- We should require that $s_0, \ldots, s_k$ are distinct
  - Otherwise there's a shorter counterexample
- And we should not allow any but s_0 to satisfy I
  - Otherwise there's a shorter counterexample
- If there's no path of length k satisfying these two constraints, and no counterexample has been found of length less than k, then we have verified P
  - By finding its finite diameter

Alternatively, Automated Induction via (Infinite or Finite) BMC

- Ordinary inductive invariance (for P):

  Basis: $I(s_0) \supset P(s_0)$

  Step: $P(r_1) \wedge T(r_1, r_2) \supset P(r_2)$

- Extend to induction of depth k:

  Basis: No counterexample of length k or less

  Step: $P(r_1) \wedge T(r_1, r_2) \wedge P(r_2) \wedge \cdots \wedge P(r_{k-1}) \wedge T(r_{k-1}, r_k) \supset P(r_k)$

  These are close relatives of the BMC formulas

- Induction for $k = 2, 3, 4 \ldots$ may succeed where $k = 1$ does not

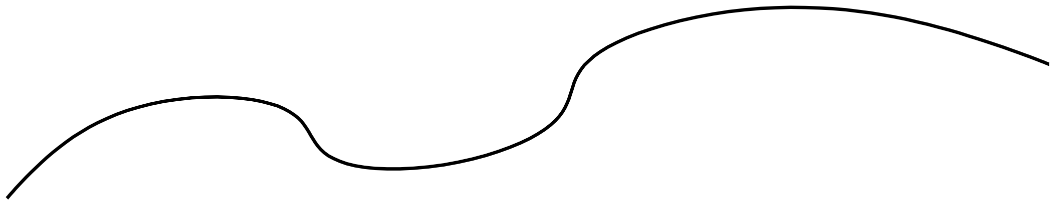
- Avoid loops and degenerate cases in the antecedent paths as in BMC

- Method is complete for some problems (e.g., timed automata)

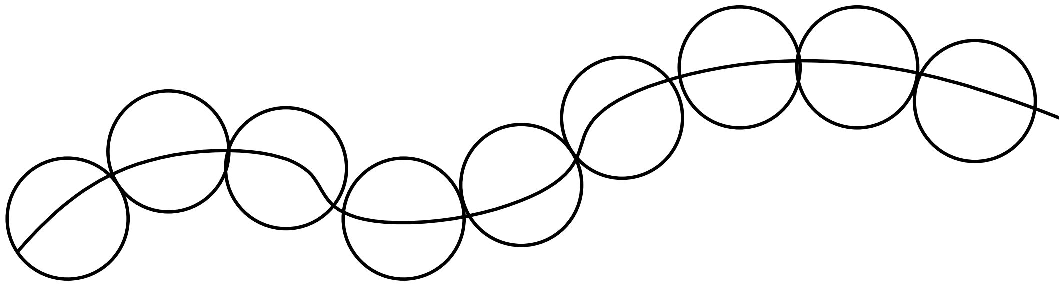
BMC Integrates With Informal Methods

- 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



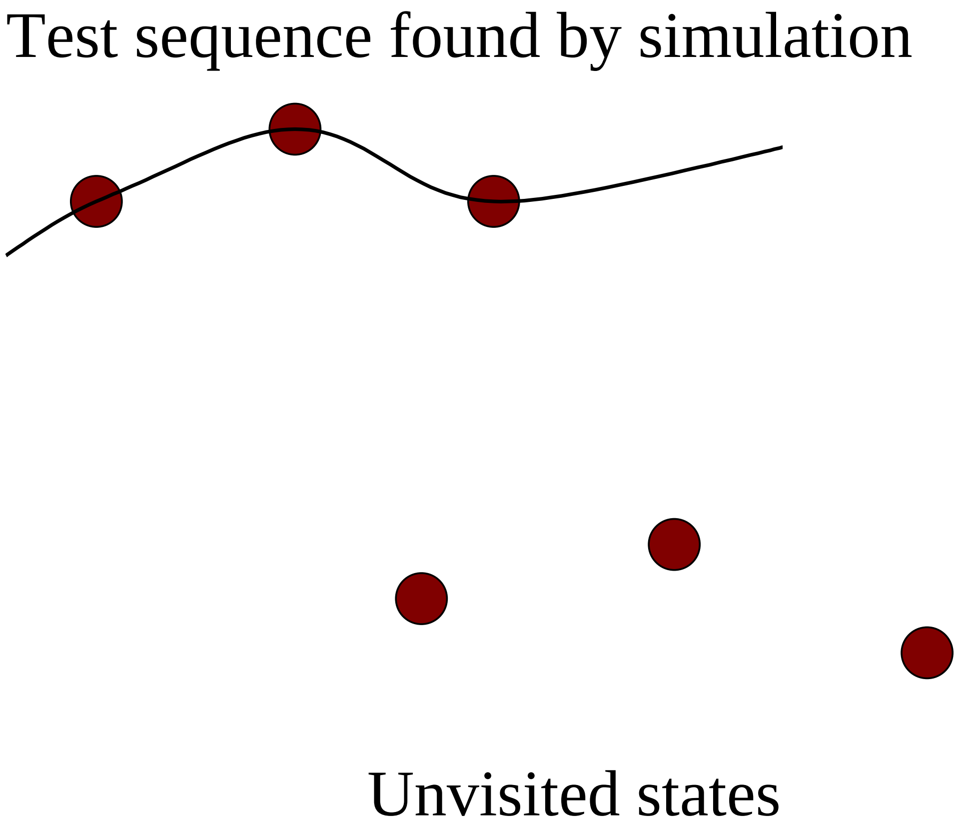
Test sequence found by simulation



Test sequence amplified by bounded model checking

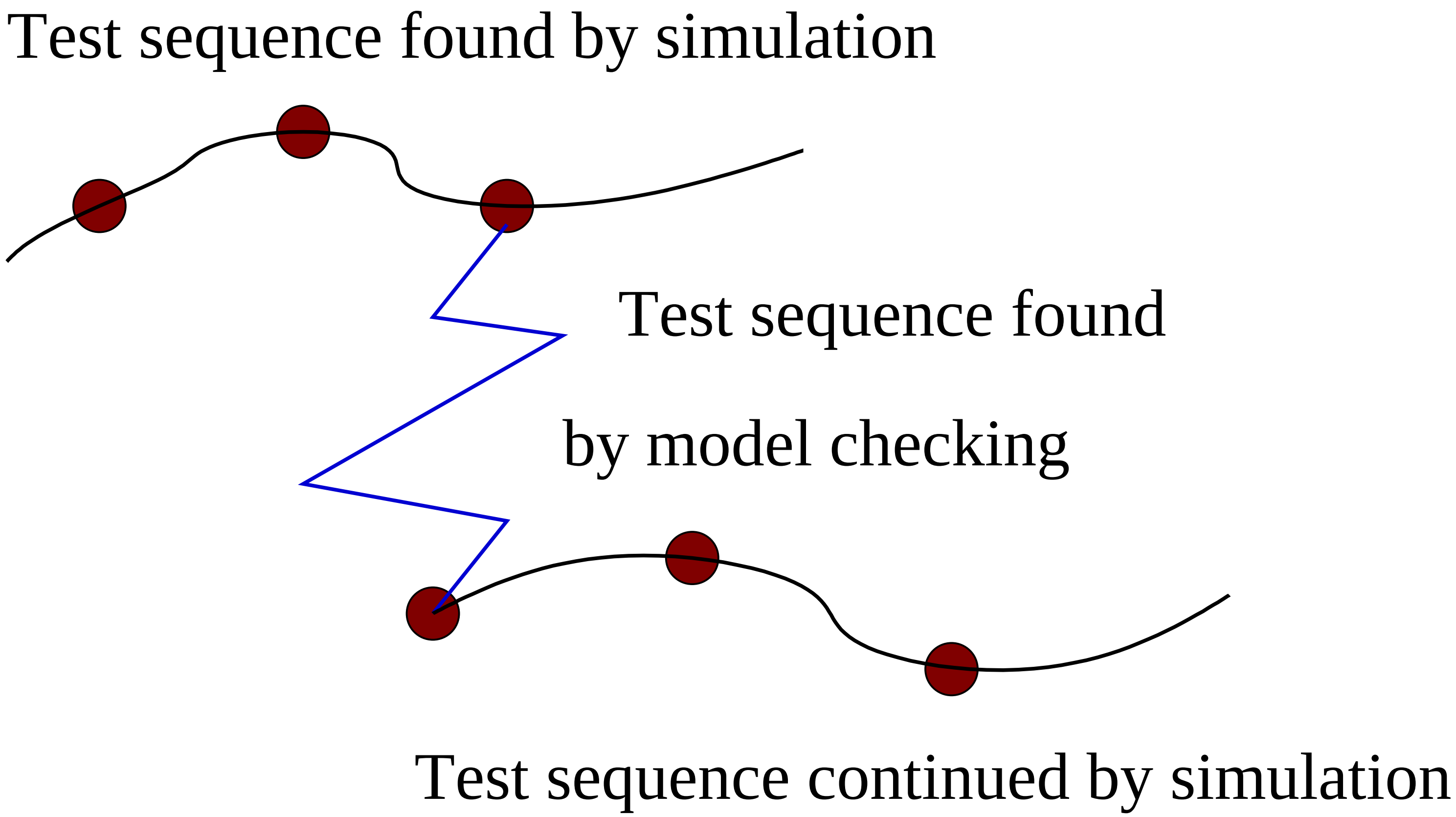
Extending The Reach Of Simulation

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



Property-Preserving Abstractions

- Beyond amplification and extension lies abstraction
- Given a transition relation T on S and property P, a property-preserving abstraction yields a transition relation $\hat{T}$ on $\hat{S}$ and property $\hat{P}$ such that

$$\hat{T} \models \hat{P} \Rightarrow T \models P$$

  Where $\hat{T}$ and $\hat{P}$ that are simple to analyze

- A good abstraction typically (for safety properties) introduces nondeterminism while preserving the property

Calculating an Abstraction

- We need to figure out if we need a transition between any pair of abstract states
- Given abstraction function $\phi : [S \rightarrow \hat{S}]$ we have

$$\hat{T}(\hat{s}_1, \hat{s}_2) \Leftrightarrow \exists s_1, s_2 : \hat{s}_1 = \phi(s_1) \wedge \hat{s}_2 = \phi(s_2) \wedge T(s_1, s_2)$$

- We use highly automated theorem proving to construct the abstracted system:
  - If we include transition iff the formula is proved
  - There's a chance we may fail to prove true formulas
  - This will produce unsound abstractions
- So turn the problem around and calculate when we don't need a transition: omit transition iff the formula is proved

$$\neg\hat{T}(\hat{s}_1, \hat{s}_2) \Leftrightarrow \vdash \forall s_1, s_2 : \hat{s}_1 \neq \phi(s_1) \vee \hat{s}_2 \neq \phi(s_2) \vee \neg T(s_1, s_2)$$

- Now theorem-proving failure affects accuracy, not soundness
- We call this "failure tolerant theorem proving"

Hybrid Abstraction

- A variant on this approach can reduce hybrid systems (e.g., Simulink/Stateflow) to sound discrete abstractions
  - Which are then examined by (either bounded or explicit state) model checking
- Abstracts polynomials over continuous variables and their first j derivatives to their qualitative signs $\{-, 0, +\}$.
- Computation uses a decision procedure over real closed fields
- The method is complete for linear hybrid systems
- Heuristically effective for others
- Allows computation of reachable states for hybrid systems (e.g., "will these two aircraft ever collide?")
- Has solved harder problems than other methods
- . . . Patent pending

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
- This is the fruit of 20 years of sustained research in the field
- 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/`

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
  - But must fit in their flow

  Can shift from technology push to pull

One Thread in a Future for Formal Methods

- Develop and sell automated analysis tools as adjuncts to those used in industry
  - Like Motorola have done
  - Or Praxis (PVL), Esterel, RSI
  - Must have very good understanding of your market segment
- Or, develop underlying engines for deductive calculation and license those to tool vendors
  - Like Prover do (e.g., in Esterel Design Verifier)
  - You need IP protection to engage here
  - And the best technology
  - And you need to work with the vendors to develop cool ways they can use your technology

A Bigger Thread: 21st Century Mathematics

- The industrialization of the 19th and 20th century was based on continuous mathematics
  - And its automation
- That of the 21st century will be based on symbolic mathematics
  - Whose automation is now feasible

  Allows analysis of systems too complex and numerically too indeterminate for classical methods
- Example: molecular biology
  - Knockouts in E.Coli (SRI; Maude)
  - Cell differentiation in C.Elegans (Weizmann; Play-in/out)
  - Delta-Notch signaling (SRI, Stanford; Hybrid SAL)
  - Sporolation in B.Subtilis (SRI; Hybrid SAL)