AIAA GNC Conference 19 August 2008, Honolulu conf center, based on
Kickoff for “Formal Verification and Automated Testing for Diagnostic and Monitoring Systems Using Hybrid Abstraction”
NASA LaRC/NIA, 29 April 2008

Formal Verification and Automated Testing For Diagnostic and Monitoring Systems

Bruno Dutertre, John Rushby, Ashish Tiwari (SRI)
César Muñoz and Radu Siminiceanu (NIA)

Computer Science Laboratory
SRI International
Menlo Park CA USA

Project Context

- New 3-year project under IVHM
  (Integrated Vehicle Health Management)
- Started January 2008
- Cooperative agreement with NASA LaRC
  - Managed by Ben DiVito
- Approximately equal split between SRI and NIA
- Describe our technology, ideas, approach, and seek your input

Technical Context

- Diagnostic and monitoring systems are elements of IVHM
- Monitor physical aspects of an aircraft for indications of problems
  - May then alert maintenance or flight crews
  - Or may try to diagnose specific fault and recommend or perform remedial action
    - ⋆ Autonomy needed for UAVs, similar to Spacecraft
- Modern developments enlarge the scope from individual aircraft to many interacting aircraft (e.g., NGATS)
- Our focus: support assurance and certification of these functions
- Specifically, formal methods for verification and test automation

Formal Analysis

- Simulations (e.g., using Matlab) examine only a tiny fraction of possible behaviors
- For assurance, we are interested in all possible behaviors
- Can sometimes achieve this using formal methods
- These are methods of calculation that use symbolic techniques
    - e.g., symbolic expression $x < y$ represents an infinite number of explicit states (0,1), (0,2), ... (1,2), (1,3)...
    - cf. universal demonstration $(x - y) \times (x + y) = (x^2 - y^2)$ vs. experiments for specific values of x and y.
- Fairly well-known for finite discrete systems
    - e.g., symbolic and bounded model checking (SMC, BMC)
- Exciting extensions to some infinite and continuous systems
    - e.g., infinite bounded model checking using SMT solvers

SAT Solving

- Find satisfying assignment to a propositional logic formula
- Formula can be represented as a set of clauses
  - In CNF: conjunction of disjunctions
  - Find an assignment of truth values to variable that makes at least one literal in each clause TRUE
  - Literal: an atomic proposition A or its negation $\bar{A}$
- Example: given following 4 clauses
  - A, B
  - C, D
  - E
  - $\bar{A}, \bar{D}, \bar{E}$

  One solution is $A, C, E, \bar{D}$

  (A, D, E is not and cannot be extended to be one)
- Do this when there are 1,000,000s of variables and clauses

SAT Solvers

- SAT solving is the quintessential NP-complete problem
- But now amazingly fast in practice (most of the time)
  - Breakthroughs (starting with Chaff) since 2001
    - ⋆ Building on earlier innovations in SATO, GRASP
  - Sustained improvements, honed by competition
- Has become a commodity technology
  - MiniSAT is 700 SLOC
- Can think of it as massively effective search
  - So use it when your problem can be formulated as SAT
- Used in bounded model checking and in AI planning
  - Routine to handle 10^{300} states

SAT Plus Theories

- SAT can encode operations and relations on bounded integers
  - Using bitvector representation
  - With adders etc. represented as Boolean circuits

  And other finite data types and structures
- But cannot do not unbounded types (e.g., reals), or infinite structures (e.g., queues, lists)
- And even bounded arithmetic can be slow when large
- There are fast decision procedures for these theories
- But their basic form works only on conjunctions
- General propositional structure requires case analysis
  - Should use efficient search strategies of SAT solvers

  That's what an SMT solver does

Decidable Theories

- Many useful theories are decidable
  (at least in their unquantified forms)
  - Equality with uninterpreted function symbols
    $x = y \wedge f(f(f(x))) = f(x) \supset f(f(f(f(f(y))))) = f(x)$
  - Function, record, and tuple updates
    $f \textbf{ with } [(x) := y](z) \stackrel{\mathrm{def}}{=} \textbf{if } z = x \textbf{ then } y \textbf{ else } f(z)$
  - Linear arithmetic (over integers and rationals)
    $x \leq y \wedge x \leq 1 - y \wedge 2 \times x \geq 1 \supset 4 \times x = 2$
  - Special (fast) case: difference logic
    $x - y < c$
- Combinations of decidable theories are (usually) decidable

$e.g.,\, 2 \times car(x) - 3 \times cdr(x) = f(cdr(x)) \supset$

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

Uses equality, uninterpreted functions, linear arithmetic, lists

SMT Solving

- SMT is Satisfiability Modulo Theories
- Individual and combined decision procedures decide conjunctions of formulas in their decided theories
- SMT allows general propositional structure
  - e.g., $(x \leq y \vee y = 5) \wedge (x < 0 \vee y \leq x) \wedge x \neq y$

    ...possibly continued for 1000s of terms
- Combines decision procedures with search strategies of modern SAT solvers
- There are several effective SMT solvers
- Ours is Yices
- Honed by competition, can handle tens of thousands of variables and constraints

Bounded Model Checking (BMC)

- Given system specified by initiality predicate I and transition relation T on states S (i.e., a nondeterministic state machine)
- Is there a counterexample to property P in k steps or less?
- Find assignment to states $s_0, \ldots, s_k$ satisfying

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

- Given a Boolean encoding of I, T, and P (i.e., circuit), this is a propositional satisfiability (SAT) problem
- But if I, T and P use decidable but unbounded types, then it's an SMT problem: infinite bounded model checking
- (Infinite) BMC also generates test cases and plans
  - State the goal as negated property

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

k-Induction

- BMC extends from refutation to verification via k-induction
  - Other ways include finding diameter of the statespace, abstraction/refinement, using interpolants to find fixpoint
- Ordinary inductive invariance (for P):

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

  Step: $P(r_0) \wedge T(r_0, r_1) \supset P(r_1)$
- Extend to induction of depth k:

  Basis: No counterexample of length k or less

  Step: $P(r_0) \wedge T(r_0, r_1) \wedge P(r_1) \wedge \cdots \wedge P(r_{k-1}) \wedge T(r_{k-1}, r_k) \supset P(r_k)$

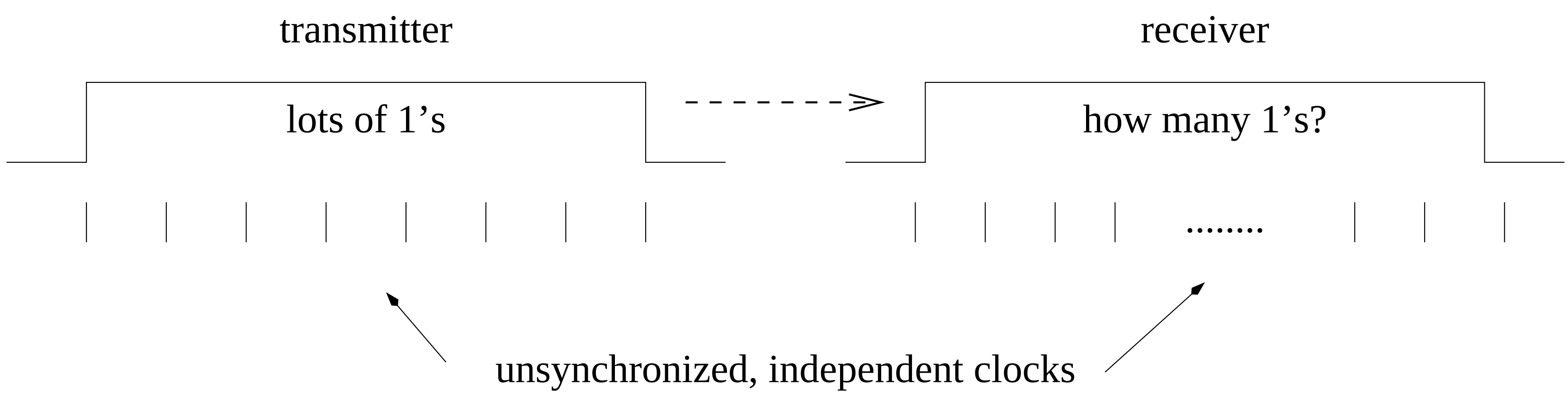
  These are close relatives of the BMC formulas
- Induction for $k = 2, 3, 4 \ldots$ may succeed where $k = 1$ does not
- Note that counterexamples help debug invariant

Application: Verification of Real Time Programs

- Continuous time excludes automation by finite state methods
- Timed automata methods (e.g., Uppaal)
  - Handle continuous time
  - But are defeated by the case explosion when (discrete) faults are considered as well
- Infinite bounded model checkers can handle both dimensions
  - With discrete time, can have a clock module that advances time one tick at a time
    - ⋆ Each module sets a timeout, waits for the clock to reach that value, then does its thing, and repeats
  - Better: move the timeout to the clock module and let it advance time all the way to the next timeout
    - ⋆ These are Timeout Automata (Dutertre and Sorea): and they work for continuous time

Example: Biphase Mark Protocol

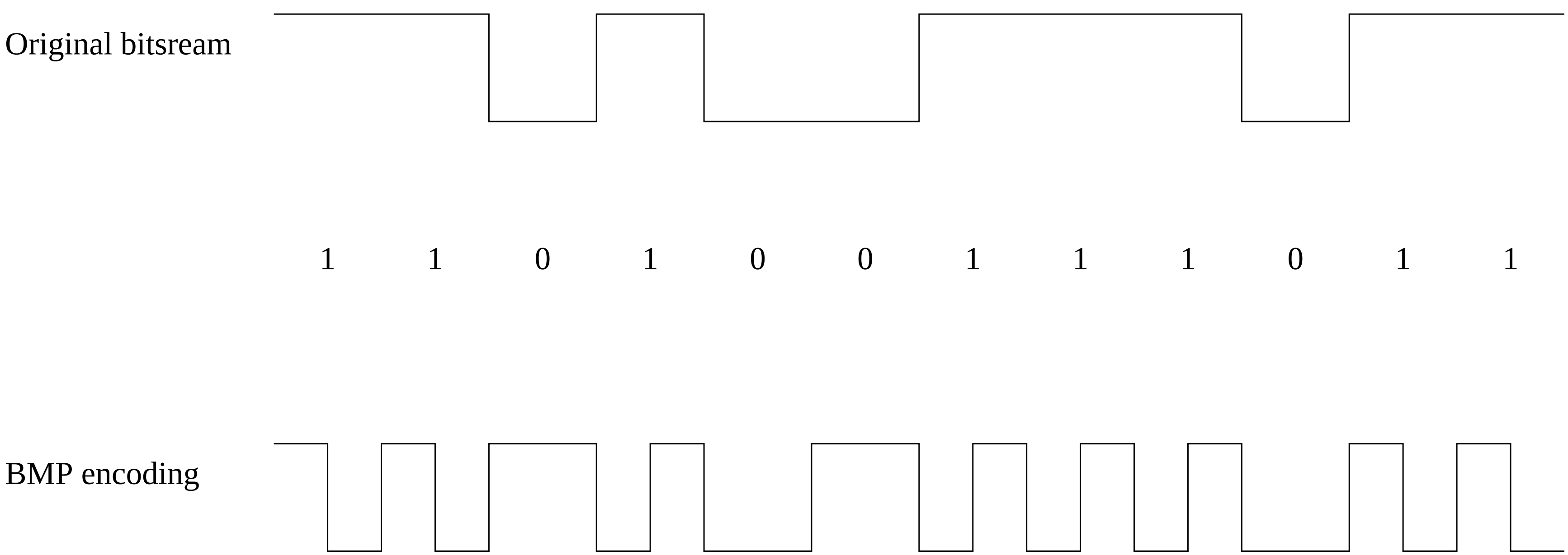
Biphase Mark is a protocol for asynchronous communication



- Clocks at either end may be skewed and have different rates, and jitter
- So have to encode a clock in the data stream
- Used in CDs, Ethernet
- Verification identifies parameter values for which data is reliably transmitted

Example: Biphase Mark Protocol (ctd)

- Flip the signal at the begining of every bit cell
- For a 1 bit, flip it in the middle, too
- For a 0 bit, leave it constant



Prove this works provided the sender and receiver clocks run at similar rates

Biphase Mark Protocol Verification

- Verified by human-guided proof in ACL2 by J Moore (1994)
- Three different verifications used PVS
  - One by Groote and Vaandrager used PVS + UPPAAL required 37 invariants, 4,000 proof steps, hours of prover time to check

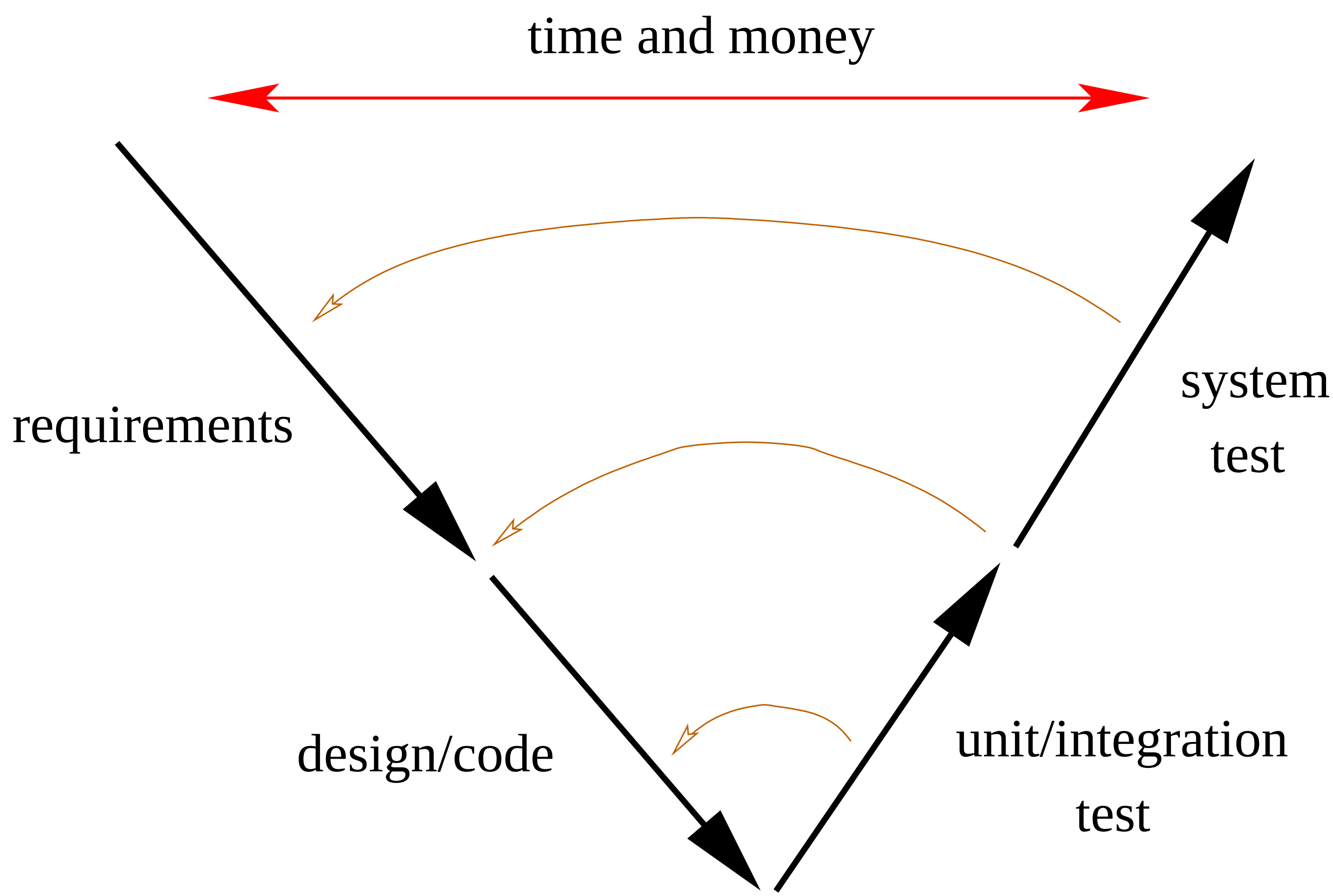
Biphase Mark Protocol Verification (ctd)

- Brown and Pike recently did it with sal-inf-bmc
  - Used timeout automata to model timed aspects
  - Statement of theorem discovered systematically using disjunctive invariants (7 disjuncts)
  - Three lemmas proved automatically with 1-induction,
  - Theorem proved automatically using 5-induction
  - Verification takes seconds to check
- Adapted verification to 8-N-1 protocol (used in UARTs)
  - Automated proofs more reusable than step-by-step ones
  - Additional lemma proved with 13-induction
  - Theorem proved with 3-induction (7 disjuncts)
  - Revealed a bug in published application note

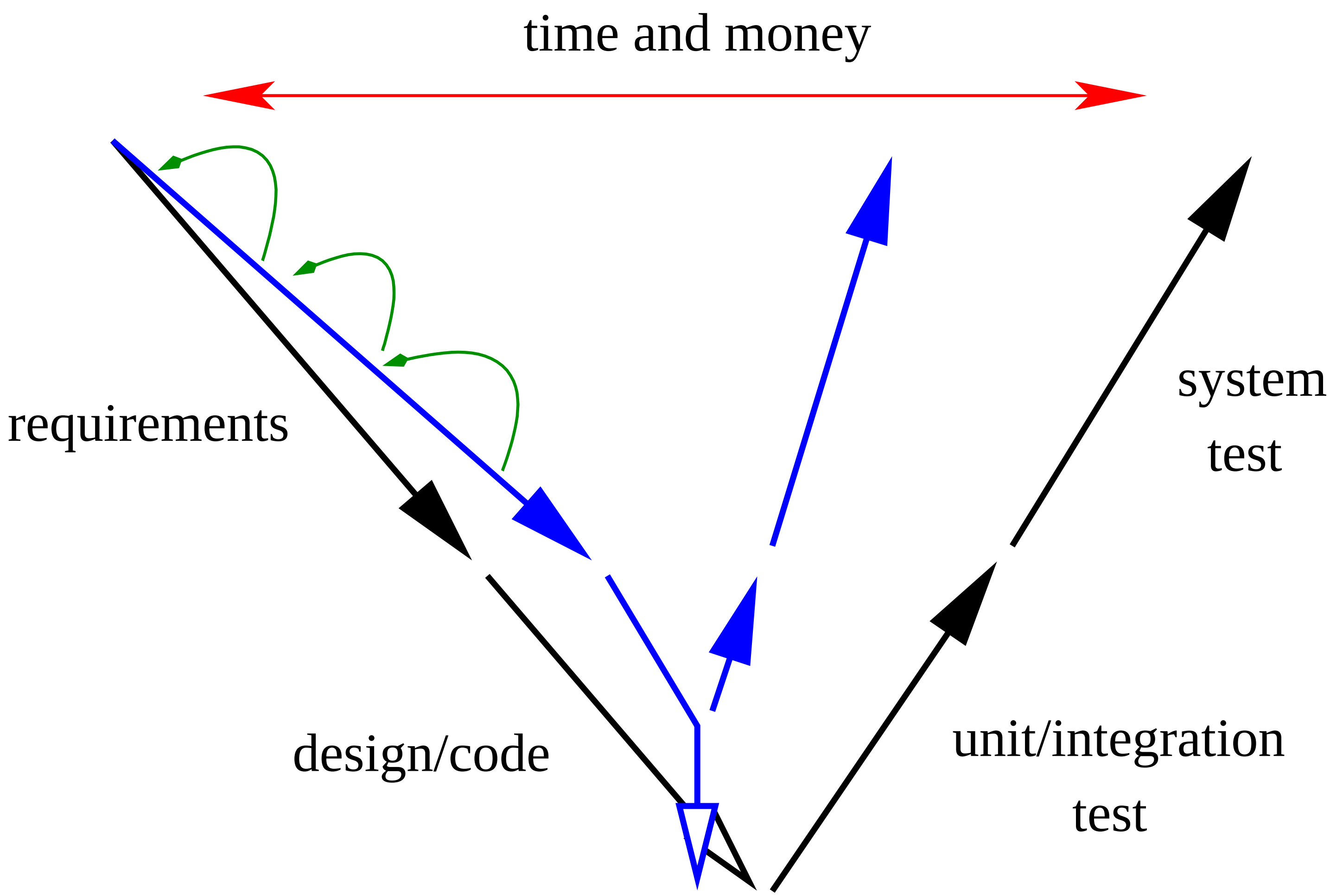
Industrial Uptake

- The move to model based development is an opportunity to move formal methods into industrial practice
- For the first time we have “machinable” artifacts prior to the code—i.e., models
- Simulink Design Verifier from Mathworks is based on exactly the technology just described
  - It does verification, refutation, and test generation for discrete time Stateflow/Simulink by k-induction and infinite bounded model checking using an SMT solver
  - Its principal developer, Gregoire Hamon, is from our group
- Tools like this can improve the quality of early-lifecycle products, reducing overall development time and cost

Traditional Vee Diagram (Much Simplified)



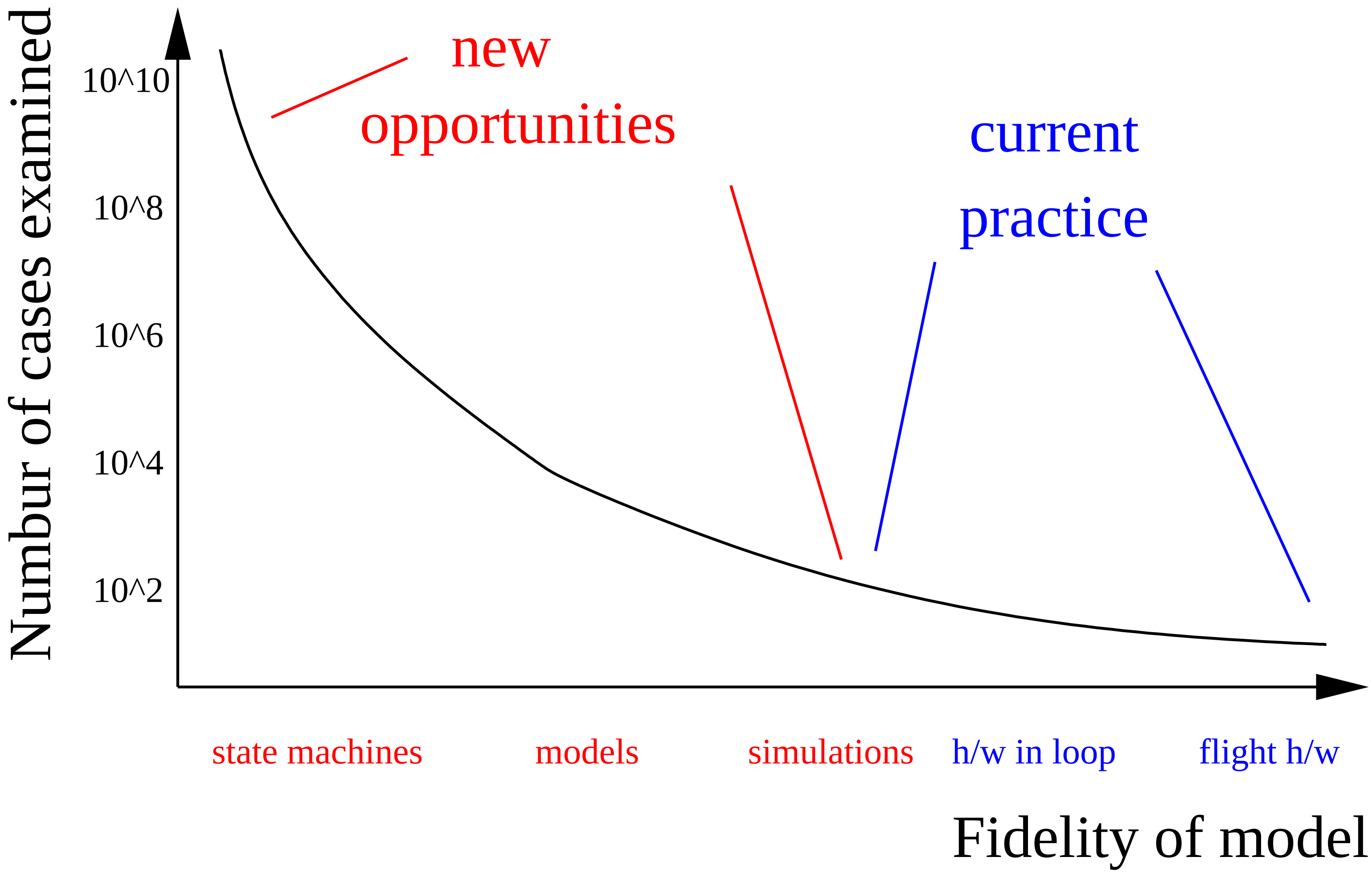
Vee Diagram Tightened with Formal Methods



Example: Rockwell-Collins

The Spectrum of V&V for Autonomous IVHM

A wealth of opportunities to the left; can apply them early, too



Diagnostic and Monitoring Systems

- These usually operate with respect to some model of the monitored system
  - Problem indicated when observed behavior of actual system departs from that of its (fault-free) model
- Models of physical systems generally involve differential equations
- Which may change as system enters different discrete modes
  - e.g., different lift, drag for different settings of the flaps
- So the models combine discrete and continuous mathematics
  - e.g., Simulink/Stateflow
- Hybrid systems (aka. hybrid automata) provide a formal framework for analyzing these models

Analysis of Hybrid Systems

- We are interested in questions such as
  - Can the system ever get into a state satisfying some relationship among the continuous variables? (e.g., the positions of two aircraft are closer than a desired minimum)
  - Can we automate operation of a testbed for the system (e.g., the actual code plus simulated hardware) to achieve desired test coverage?
- Both these are solved if we can analyze hybrid systems for invariants (or, more generally, safety properties)
  - Test cases are derived from counterexamples
- Note, control theory and infinite-BMC do not solve these

Formal Analysis of Hybrid Systems

- Analysis of hybrid systems is a challenging problem
- Little progress since HyTech of 1995
- Tools can seldom handle more than 5 continuous variables
- This is because they work by calculating the reachable states
- Overkill: we just want to know if a specific property is reachable
- Hybrid Abstraction (Tiwari and others 2002–) focuses on this and is much more efficient
- Can often handle 15 or 25 continuous variables
- Automated by HybridSAL: freely distributed since 2007
- Trivial example in the paper

Hybrid Abstraction

- Hybrid Abstraction constructs a discrete overapproximation (abstraction) to the hybrid system
- Can then analyze the abstraction with conventional model checker
- Overapproximation: any safety property true in the abstraction is true of the original hybrid system
- But we may be unable to prove some true properties if the abstraction is too coarse
- Dually, some test-cases derived from the model may be infeasible in the real system if the abstraction is too coarse
- Can often refine the abstraction in these cases

Hybrid Abstraction (ctd.)

- The abstraction is not a simple discretization
- It replaces the continuous variables by qualitative signs of selected polynomials over the variables and their derivatives
- Qualitative signs: replace real values by {neg, zero, pos}
- Approximation is calculated by automated theorem proving over real closed fields (hard problem)
- Quality of the approximation is determined by choice of how many derivatives to consider
- And which polynomials to use
- Automated selection of polynomials (uses eigenvectors) makes the method complete for linear hybrid automata
  - Heuristically effective for others

Test Case Generation

- Generating unit test cases with a model checker is well understood
- Counterexamples to negation of desired test target provide the test cases
- Automated by SAL-ATG, a standard part of SAL
- It is also known how to integrate concrete and symbolic (concolic) execution, or constraint solving and deduction, to achieve strong coverage in domains that are hard for SMT solvers
- But for IVHM we need to test with (real or simulated) hardware in the loop

Test Case Generation with Hardware in the Loop

- Have partial control of the system (e.g., can inject a fault into the simulated hardware, supply sensor inputs)
- But cannot easily drive execution toward a specific test target
  - Uncontrolled inputs may take us away
- So test generation is this context has to be seen as synthesizing an active tester, rather than a static set of test inputs
- Hence, controller synthesis is the appropriate framework

Test Generation by Controller Synthesis

- Synthesis works by calculating the possible moves of the environment, then choosing inputs that avoid bad outcomes (basically a state exploration exercise)
- Effectiveness depends on the quality of the models of the uncontrolled parts of the system
  - e.g., the simulated physical plant
- We want a high-quality discrete over-approximation
  - May sometimes go wrong because the actual hardware cannot make a move predicted by the approximation (but better than being surprised by unexpected moves)
- Aha! Hybrid abstraction does this
- Use it to develop test automation for hardware in the loop

Current and Planned Activity

- Mostly technology development
- Developing efficient decision procedure for nonlinear arithmetic
  - Needed for infinite-BMC, calculating hybrid abstractions
  - Will also improve interactive verification (e.g., with PVS)
- A parallel project has developed a static analysis procedure for hybrid systems
  - Gulwani and Tiwari, CAV 08
  - Discovers invariants, rather than check them

  Needs the same improved decision procedures
- Developing methods for generating helper invariants
- Applying our new methods to previous analysis of operational concept for SATS airport procedures