NSA HCSS meeting, BWI Marriott 1-3 April 2003

Embedded Deduction With ICS

Leonardo de Moura, Harald Rueß,
John Rushby, and Natarajan Shankar

Computer Science Laboratory
SRI International
Menlo Park, California, USA

Embedded Deduction

- The techniques of automated deduction
  - Theorem proving, model checking

  Underpin tools for high confidence software and systems assurance based on formal methods

- Usually, the deductive capability is explicit and visible to the user

- Indeed, the tools are often identified with their deductive core
  - The SMV model checker
  - The ACL2 theorem prover

- But there's a new opportunity:
  - Provide a deductive service

  That can be embedded in other tools

Traditional Formal Methods, And Their Problems

- Use a tool with heavyweight deductive capabilities
- Extract a suitable description of the system and cast it into the specification language of the tool
  - Industry has not adopted these languages, so always some translation and interpretation involved
    - ⋆ High Level descriptions often informal, so formalization requires a lot of skill
    - ⋆ Low level descriptions (i.e., code) are formal, but large, and few errors are introduced at this level
- Express properties to be examined in the language of the tool
  - Same problems as above
- Guide the tool
  - Requires arcane skill

More Problems With Traditional Formal Methods

Not integrated with the traditional development processes

- Hard to keep the “formal shadow” current with the real system
- Oriented toward after-the-fact verification, rather than in-the-loop exploration and debugging
- Does nothing to reduce cost of existing processes such as testing
- Formal evidence does not integrate with traditional measures
  - E.g., what “coverage” does a set of properties provide?
  - How to combine evidence from model checking with that from testing?

There's An Opportunity To Fix All This

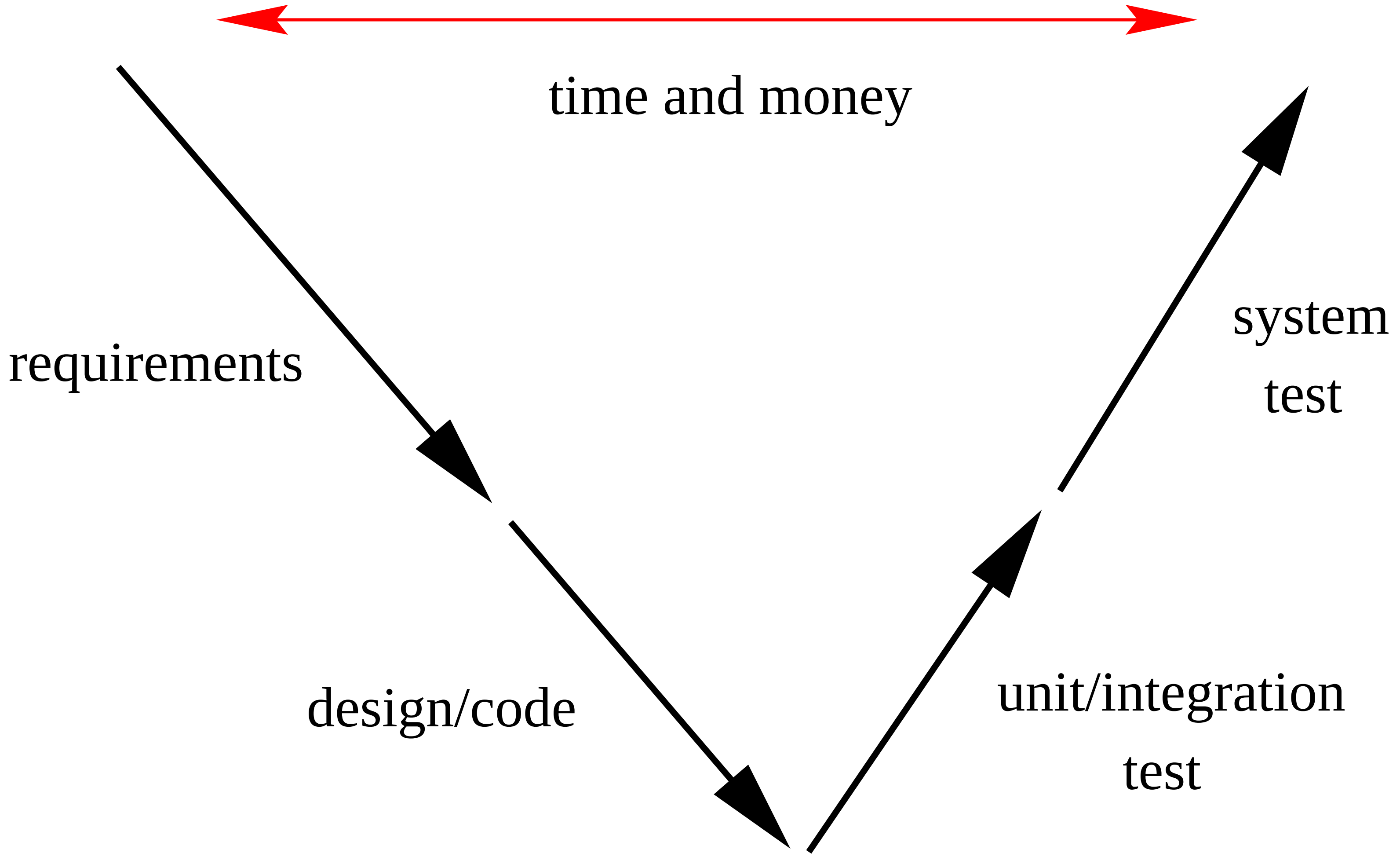
Due to a convergence of three trends

- 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, Esterel/SCADE, UML
  - Notations are not ideal for FM, but they'll do
- New kinds of formal activities
  - Fault tree analysis, test case generation, extended static checking, runtime verification, environment synthesis
- More powerful, more automated deductive techniques
  - Approaches based on "little engines of proof"
  - New engines: SAT, Multi-Shostak, "lemmas on demand"
  - New techniques: BMC, k-induction, abstraction

Nontraditional Formal Methods

- Embed formal analyses in model-based development environments
- Gives access to formal descriptions throughout lifecycle
  - That will be maintained
- Focus the analyses on providing feedback and assurance in the early lifecycle
  - Symbolic simulation, ESC, model checking
  - Provides radical improvement in traditional “V”
- And on eliminating costly and error-prone manual processes
  - Unit-level test case generation
  - Provides another radical improvement in traditional “V”

Simplified Vee Diagram

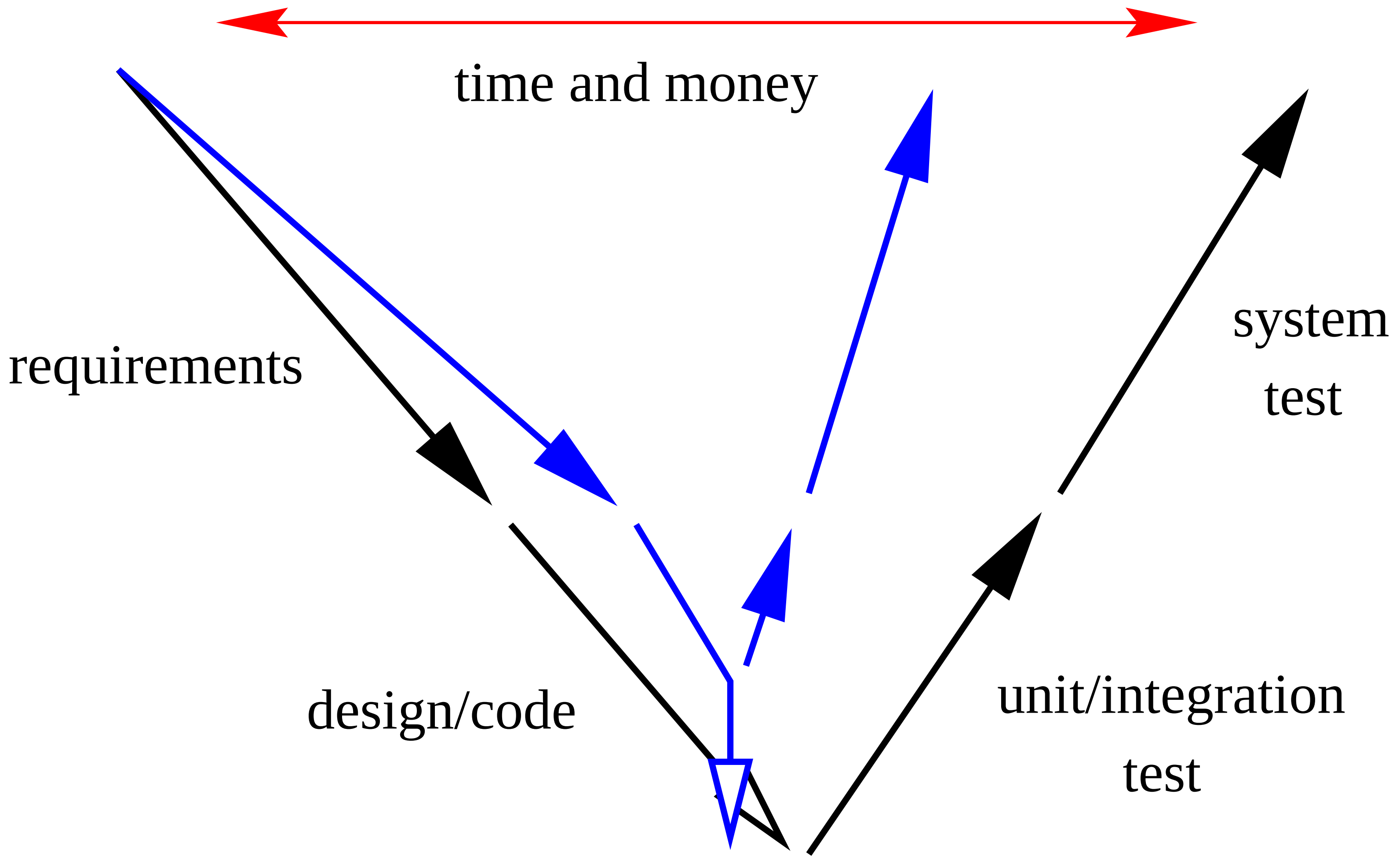


Hope is that embedded formal methods can tighten the vee

Tightening the Vee

- If formal methods could reduce requirements faults, we would reduce the amount of rework, and steepen both sides
- If formal methods could automate some testing procedures, would steepen right side (especially bottom right)
- If formal methods could automate coding, would steepen bottom left
- If formal methods could eliminate some testing procedures, would further steepen bottom right

Tighted Vee Diagram



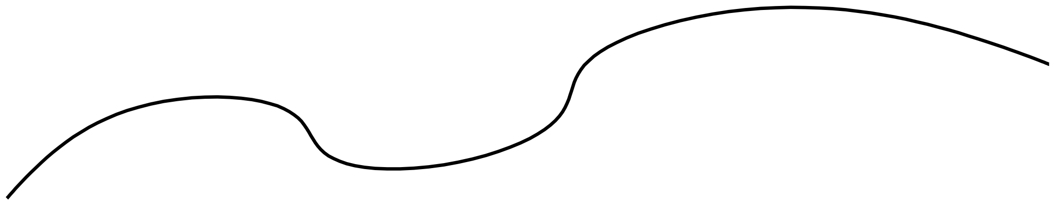
Bounded Model Checking

- A key technology that finds many applications in tightening the Vee is bounded model checking (BMC)
- Is there a counterexample of length k to this property?
- Same method generates structural testcases (counterexample to “there’s no execution that takes this path”)
- Try $k = 1, 2, \ldots 100 \ldots$ until you find a bug or run out of resources or patience
- Needs less tinkering than BDD-based symbolic model checking, and can handle bigger systems and find deeper bugs
- Now widely used in hardware verification

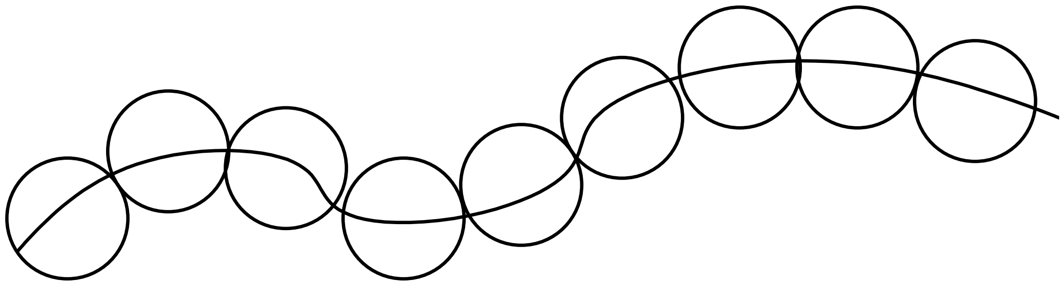
BMC Integrates With Informal Methods

- Hardware designs can have millions of state bits and interesting traces thousands of steps long
- BMC can explore 10–100 steps on hundreds of state bits
- So BMC doesn't get you very far from a start state
- So, instead, do it from states found during random simulation
- Can be seen as a way to "fatten" thin traces explored by simulation
  - Or to amplify the power of simulation

Amplifying The Power Of Simulation



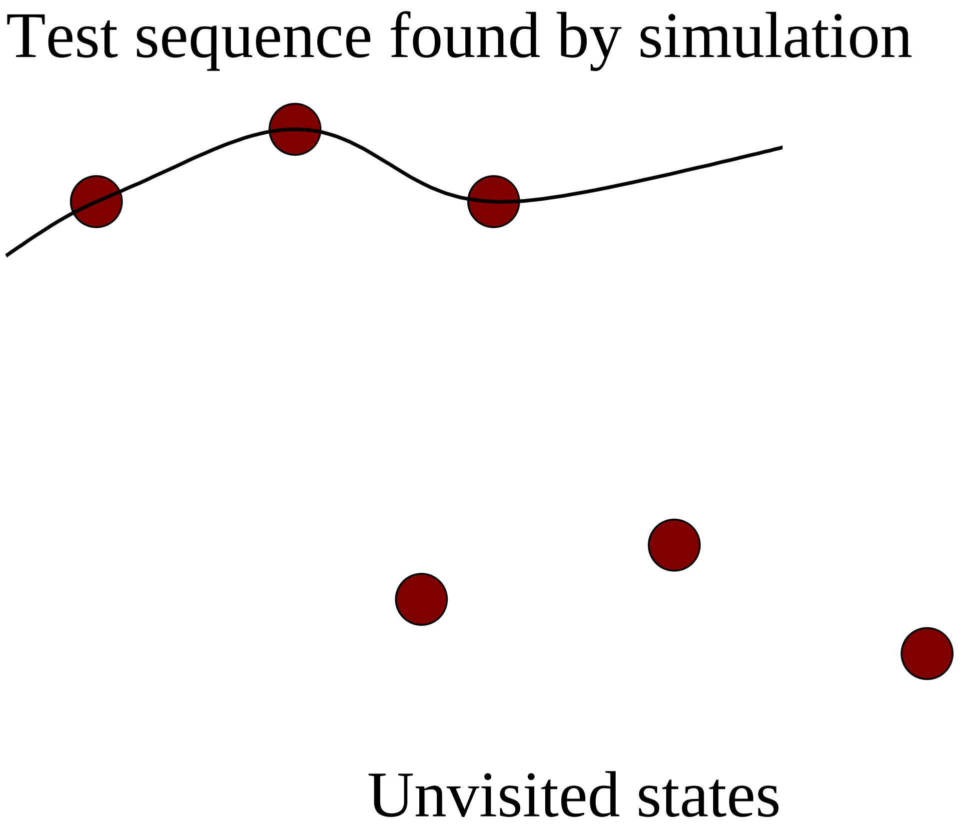
Test sequence found by simulation



Test sequence amplified by bounded model checking

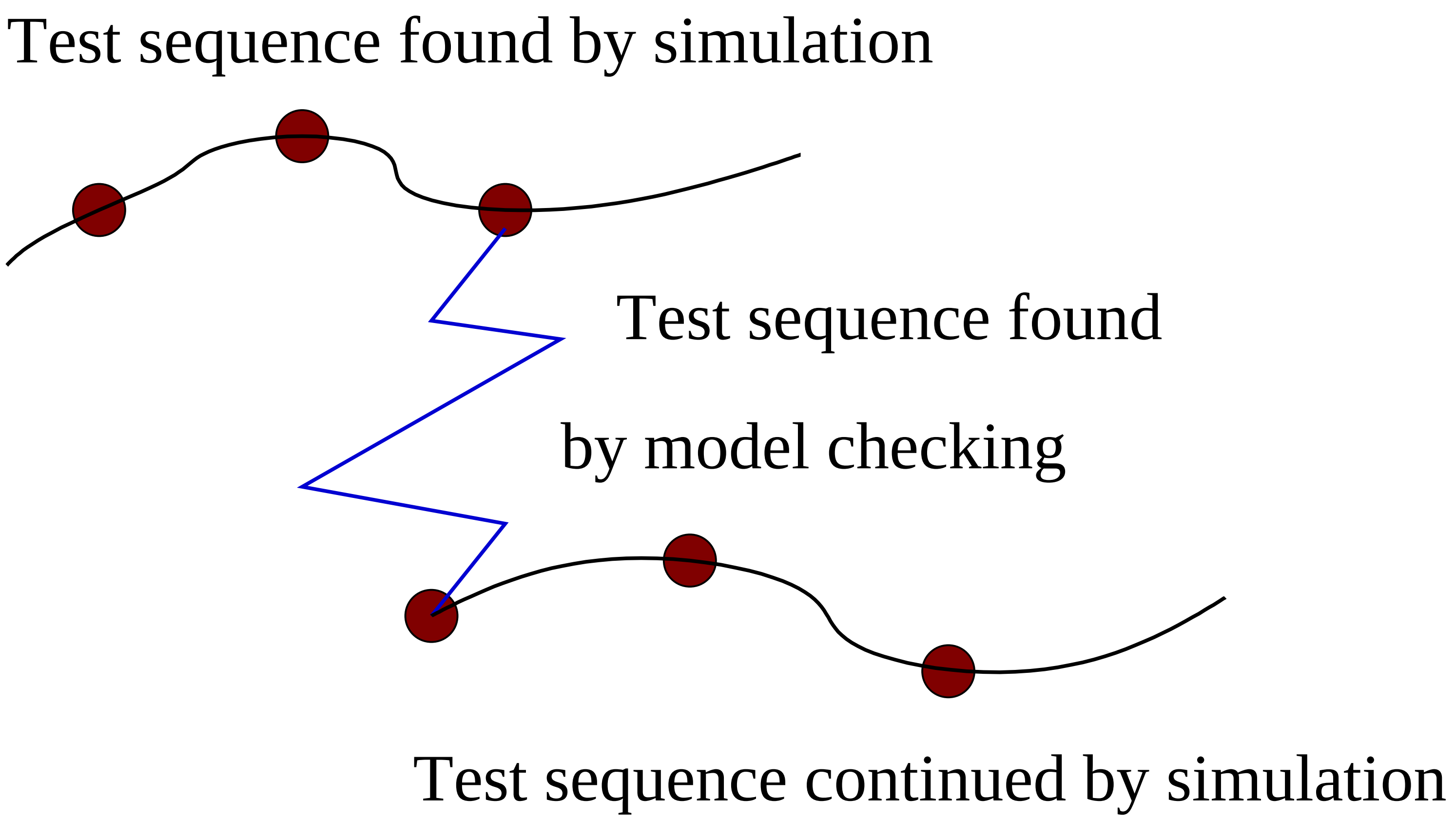
Extending The Reach Of Simulation

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 (also use abstraction to see if states are unreachable)



Bounded Model Checking (ctd.)

- Given a system specified by initiality predicate I and transition relation T, 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
- SAT solvers have become amazingly fast recently

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

- Simple example from Esterel:
  - T has 1,770 variables, formula is 4,000 lines of text
  - Want to do BMC to depth 40
  - Will have a vast, propositionally complex formula over interpreted theories

Extending (Infinite) 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

Alternatively,... Automated Induction via (Infinite) 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:

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

  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
- Need to avoid loops and degenerate cases in the antecedent paths in the same way as BMC
- Method is complete for some problems

  (e.g., timed automata)

Solving Infinite BMC Formulas

- The formulas that arise in these examples are huge
- Some examples from Esterel:
  - 227,108 terms, text representation 5Mb
  - 105,844 terms and a 3Mb text file
  - 72,291 terms and 2Mb text file
- This is the scale of problem ICS is designed to solve

What is ICS?

A modular, integrated **decision procedure** for a combination of theories that is sound and complete

The name comes from Integrated Canonizer/Solver

Decision procedure: tells 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?

Uses heuristics for speed, but always terminates and gives the correct answer

What is ICS? Continued 1...

A modular, integrated decision procedure for a **combination of theories** that is sound and complete

Combination of theories: 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

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

What is ICS? Continued 2...

A **modular**, **integrated** decision procedure for a combination of theories that is **sound and complete**

Modular: the combined decision procedure is built from decision procedures for the individual theories

Integrated: but they are tightly integrated for speed

Sound and Complete: other approaches (e.g., Nelson-Oppen integration) are significantly slower; approaches based on original Shostak method are incomplete and nonterminating

We have formally verified the correct integration

And previous generalizations were unsound (combination of solvers may not be a solver)

We've corrected this, too

What is ICS? Continued 3...

Or whether one formula is a consequence of others...

Rich API: for many applications, do not want just a one-shot decision (does this follow from that?) but want to explore alternatives (suppose I retract this and assert that, does it still follow?)

ICS uses very efficient data structures that allow assertion and retraction for general proof search

What is ICS? Continued 4. . .

ICS decides the combination of. . . and **propositional calculus**

Propositional Calculus: ICS includes a state-of-the-art SAT solver that is tightly integrated with the core decision procedures so that it can solve huge propositional combinations of formulas over decided terms

Can think of ICS as a SAT solver where the terms are not restricted to Booleans, but can be arithmetic and other decidable expressions

SAT Solving in ICS

- Idea is to extend the efficient search of a modern SAT solver to propositionally complex formulas with interpreted terms at the leaves
  - E.g., $x < y \wedge (f(x) = y \vee 2 * g(y) < \epsilon) \vee \ldots$ for thousands of terms
- Replace the terms by propositional variables
- Get a solution from the SAT solver (if none, we are done)
- Restore the interpretation of variables and send the conjunction to the ICS core decision procedure
- If satisfiable, we are done
- If not, ask SAT solver for a new assignment

Solving in ICS (ctd.)

- But 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
- 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 $x < 3 \wedge 2x \geq 5$
  - Add lemma $\neg(p \wedge q)$ to SAT problem
  - SAT solver then suggests $r = T$
  - Interpret as $x = 4$ and we are done

ICS Compared With a SAT Solver

Can encode some datatypes in SAT

- E.g., bitvectors, bounded integers (as a bitvector)

Then provide SAT-level implementations of operations on them

- E.g., hardware-like adders, shifters

And that will semi-decide some combination of theories

Doesn't work for real arithmetic (cf., Airbus A340-500 experiments with SCADE and Prover, and Honeywell voter experiments), where true (unbounded) integers or other important theories decided by ICS are present

Exponentially less efficient than ICS on many things where it does work (e.g., barrel shifter)

Why Not Just Combine A Decision Procedure With An OTS SAT Solver?

- That's where we started
- The intense iteration between the SAT solver and the decision procedure requires each to maintain state and to do restart/backup very efficiently
- Also, we want "don't cares" (get in the way of standard heuristics)
- The SAT solver in ICS is tuned to the task, and yields several orders of magnitude improvement over a looser integration with an OTS solver
- As a pure SAT solver, ICS is competitive with Chaff

What is ICS? Continued 5...

- Core ICS is implemented in Objective Caml
- Its SAT solver is 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 has been ported to MAC OS X and to Windows XP (under cygwin)
- Freely available under license to SRI
- `ics.csl.sri.com` or `ICanSolve.com`

Applications of ICS

- ICS is intended for totally automatic operation
- So need applications having problems that can be reduced to the theories that it decides
- Like Infinite BMC, ESC, automated abstraction,. . .
- Will generally require some "glue logic"
  - Skolemization
  - Definition expansion
  - Rewriting
  - Quantifier instantiation

Glue Logic

- ICS can be used in place of the legacy decision procedures in PVS, so can use PVS to prototype glue logic
- If the glue logic uses incomplete heuristics (e.g., quantifier instantiation), then need an application that can tolerate deductive failure
  - E.g., Extended Static Checking (ESC):
    failure causes spurious warnings
  - Or automated abstractions:
    failure causes spurious counterexamples

Property-Preserving Abstractions

- Given a transition relation G on S and property P, a property-preserving abstraction yields a transition relation $\hat{G}$ on $\hat{S}$ and property $\hat{P}$ such that

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

  Where $\hat{G}$ and $\hat{P}$ that are simple to analyze (e.g., finite state)

- 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

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

- We use highly automated theorem proving to construct the abstracted system: 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

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

- Now theorem-proving failure affects accuracy, not soundness

Plans

- Currently optimizing ICS capabilities to support the deductive requirements of the Destiny system
- Later, extend to quantified theories, e.g., full Presburger
  - Undecidable in combination with uninterpreted function symbols
  - But the circumstances that trigger undecidability are sharply defined, and rare in practice
- Similarly extend to decidable fragments of nonlinear arithmetic
- Build in some glue logic capabilities (Skolemization, definition expansion, rewriting)

Looking Further Ahead

- Want a Formal Tool Bus (FTB)
- A scriptable environment in which tailored formal analyses can quickly be constructed
- Example workflow:
    - Take a hybrid system from Simulink/Stateflow, abstract it to a discrete system using deduction over real closed fields, model check it, and concretize any counterexamples and play them through Matlab; refine the abstraction and repeat

And Even Further

- A workflow is more than a tool chain
  - One analysis may be contingent on another
  - E.g., abstraction depends on an invariant

  So it is actually more like a proof:
  - FTB must do evidence management
- Formal evidence does not integrate with traditional measures
  - E.g., what “coverage” does a set of properties provide?
  - How to combine evidence from model checking with that from testing?
- Extend to approximate forms of evidential reasoning
  - Dempster-Schaefer or Bayesian Belief Networks

  And you may have what you need to develop “safety cases”