Introduction to SMT Solving And Infinite Bounded Model Checking

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, California, USA

Since The Beginning...

- Theorem provers (or, at least, proof checkers) have been a central element in mechanized verification from the first systems of King (1969) and Good (1970)
- They've evolved over the years and become specialized for this application
- With decision procedures and other automation for arithmetic, data structures, recursively and inductively defined functions and relations etc.

Until. . .

- . . . the present?
- Most significant verifications were accomplished with a theorem prover (ACL2, HOL, Isabelle, PVS. . .)

Then Along Came Model Checking

- Initially, these were just explicit state reachability analyzers
- Then BDDs and CTL
- But still finite state
- So ad-hoc downscaling required
- OK for debugging, not verification

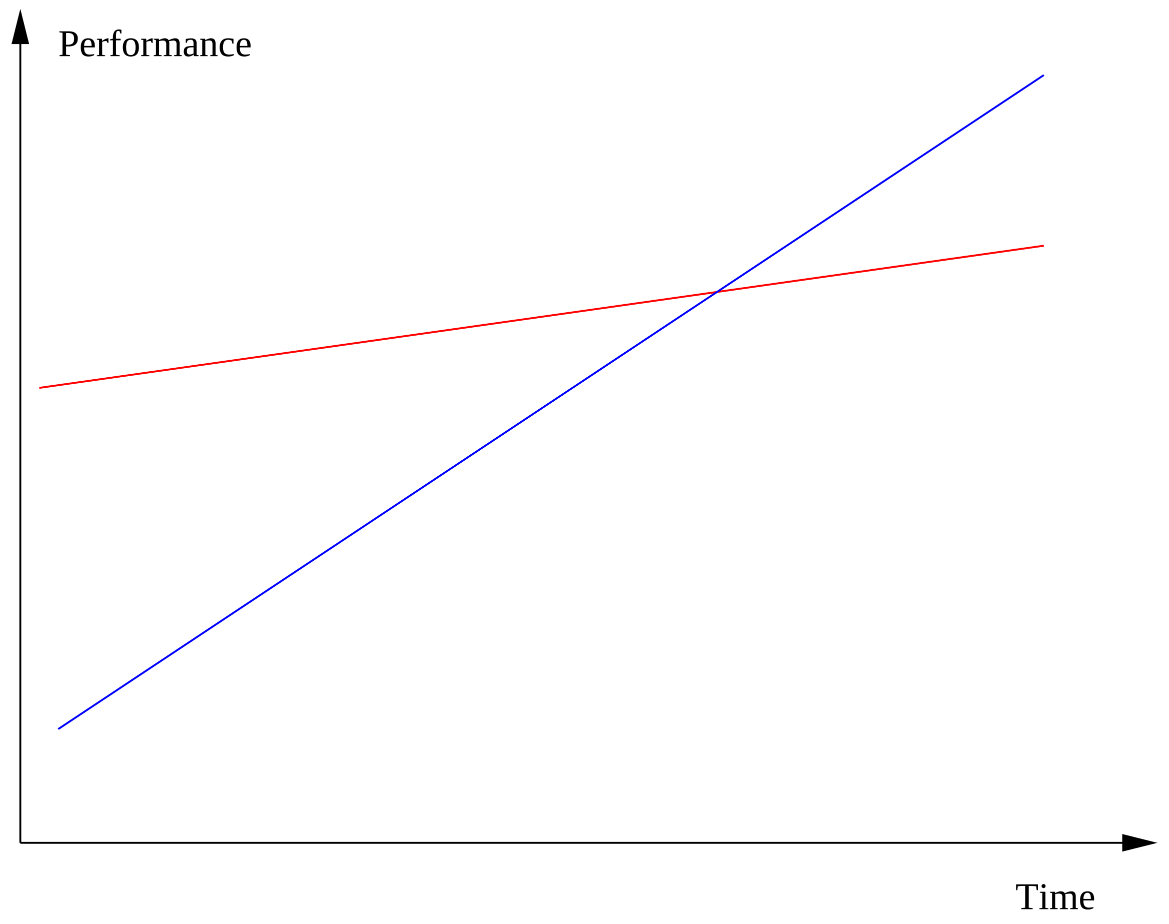
- Same for static analysis

. . . And Automated Abstraction

- Predicate abstraction provides an automatable way to construct property preserving abstractions
- And spurious counterexamples can be mined to refine inadequate abstractions (CEGAR)

- Model checking starts to encroach on the space occupied by theorem proving

Disruptive Innovation



Low-end disruption is when low-end technology overtakes the performance of high-end (Christensen)

SMT Solvers: Disruptive Innovation in Theorem Proving

- SMT stands for Satisfiability Modulo Theories
- SMT solvers extend decision procedures with the ability to handle arbitrary propositional structure
  - Traditionally, case analysis is handled heuristically in the theorem prover front end
    - ⋆ Have to be careful to avoid case explosion
  - SMT solvers use the brute force of modern SAT solving
- Or, dually, they generalize SAT solving by adding the ability to handle arithmetic and other decidable theories
- Application to verification
  - Via bounded model checking and k-induction

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
    $f \textbf{ with } [(x) := y](z) \stackrel{\text{def}}{=} \textbf{if } z = x \textbf{ then } y \textbf{ else } f(z)$
  - Linear arithmetic (over integers and rationals)
    $x \le y \wedge x \le 1 - y \wedge 2 \times x \ge 1 \supset 4 \times x = 2$
  - 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

- 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
- Should exploit search strategies of modern SAT solvers
- So replace the terms by propositional variables
  - i.e., $(A \vee B) \wedge (C \vee D) \wedge E$
- Get a solution from a SAT solver (if none, we are done)
  - e.g., A, D, E
- Restore the interpretation of variables and send the conjunction to the core decision procedure
  - i.e., $x \leq y \wedge y \leq x \wedge x \neq y$

SMT Solving by "Lemmas On Demand"

- If satisfiable, we are done
- If not, ask SAT solver for a new assignment
- But isn't it expensive to keep doing this?
- Yes, so first, do a little bit of work to find fragments that explain the unsatisfiability, and send these back to the SAT solver as additional constraints (i.e., lemmas)
  - $A \wedge D \supset \bar{E}$ (equivalently, $\bar{A} \vee \bar{D} \vee \bar{E}$)
- Iterate to termination
  - e.g., $A, C, E, \bar{D}$
  - i.e., $x \le y, x < 0, x \ne y, y \not\le x$ (simplifies to $x < y, x < 0$)
  - A satisfying assignment is $x = -3, y = 1$
- This is called "lemmas on demand" (de Moura, Ruess, Sorea) or "DPLL(T)"; it yields effective SMT solvers

Fast SMT Solvers

- There are several effective SMT solvers
  - Our ICS was among the first (released 2002)
    - ⋆ Precursors include CVC, LPSAT, Simplify...
  - Now replaced by Yices (released 2006)
  - European examples: Barcelogic, MathSAT
- Yices decides formulas in the combined theories of: linear arithmetic over integers and reals (including mixed forms), fixed size bitvectors, equality with uninterpreted functions, recursive datatypes (such as lists and trees), extensional arrays, dependently typed tuples and records of all these, lambda expressions, and some quantified formulas
- SMT solvers are being honed by competition
  - Provoked by our benchmarking in 2004
  - Now institutionalized as part of CAV, FLoC

SMT Competition

- Various divisions (depending on the theories considered)
  - Equality and uninterpreted functions
  - Difference logic $(x - y < c)$
  - Full linear arithmetic
    - ⋆ For integers as well as reals
  - Extensional arrays, bitvectors, quantification . . . etc.
- ICS was the most uniformly effective in 2004
- Yices 0.2 and Simplics (prototypes for Yices 1.0) won the advanced divisions in 2005, came second to Barcelogic in all the others
- Yices 1.0 won all 11 divisions in 2006
- Yices 1.0.10 won 6 of 12 divisions in 2007
- Yices 2.0 under development

Bounded Model Checking (BMC)

- Given system specified by initiality predicate I and transition relation T on states S
- 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

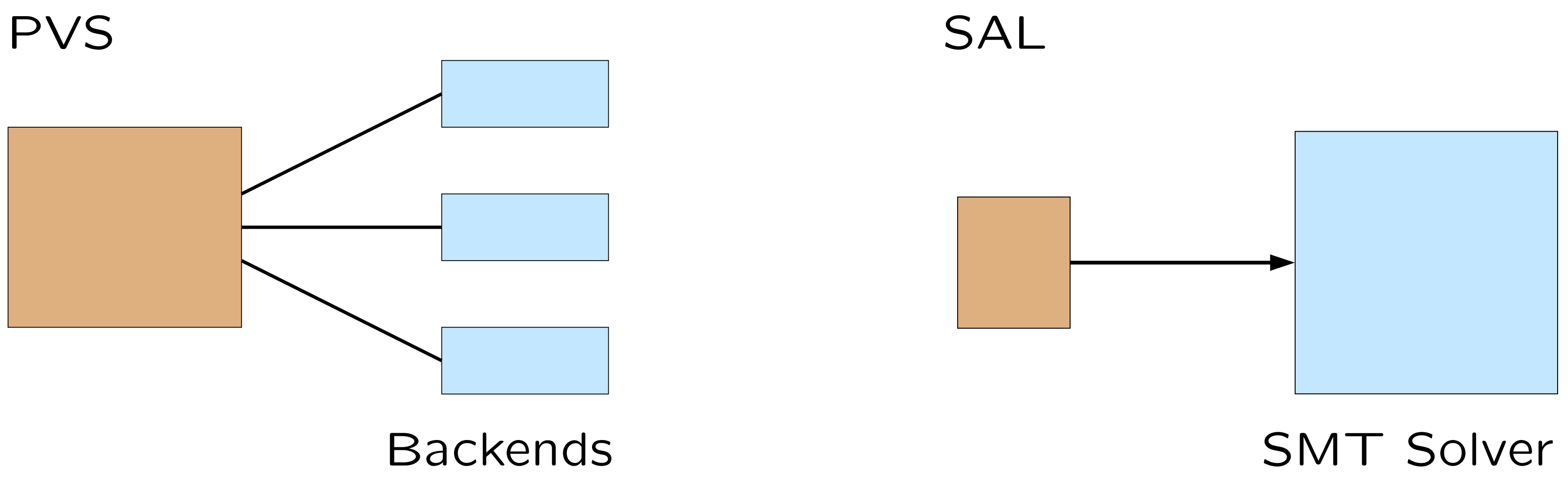
  Step: $P(r_0) \wedge T(r_0, r_1) \wedge P(r_1) \wedge \cdots \wedge P(r_{k-1}) \wedge T(r_{k-1}, r_k) \supset P(r_k)$

  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

Evolution of SMT-Based Model Checkers

- Replace the backend decision procedures of a verification system with an SMT solver, and specialize and shrink the higher-level proof manager
- Example:
  - SAL language has a type system similar to PVS, but is specialized for specification of state machines
    (as finite- or infinite-state transition relations)
  - The SAL infinite-state bounded model checker uses an SMT solver (Yices), so handles specifications over reals and integers, uninterpreted functions, etc.
  - Often used as a model checker (i.e., for refutation)
  - But can perform verification with a single higher level proof rule: k-induction (with lemmas)

Verification Systems vs. SMT-Based Model Checkers



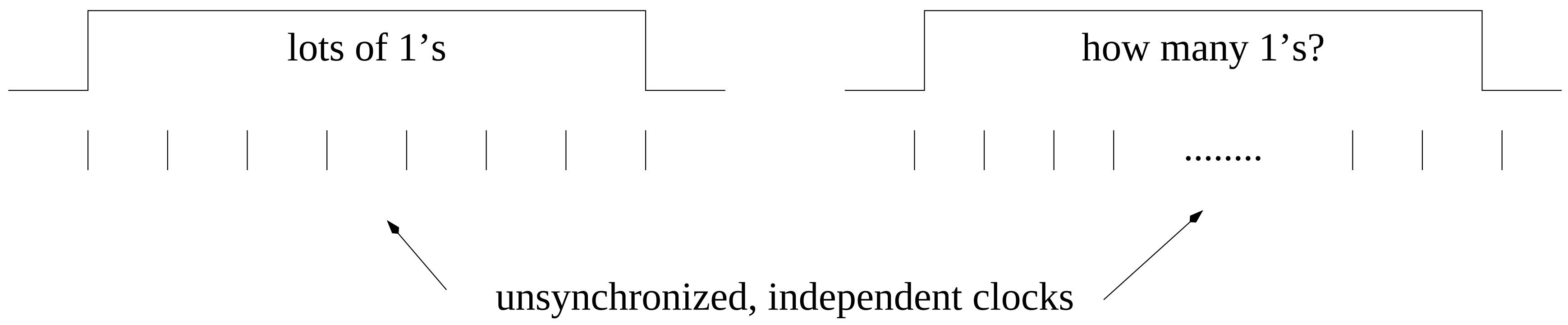
Actually, both kinds will coexist as part of the evidential tool bus—another talk

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
- SMT solvers 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

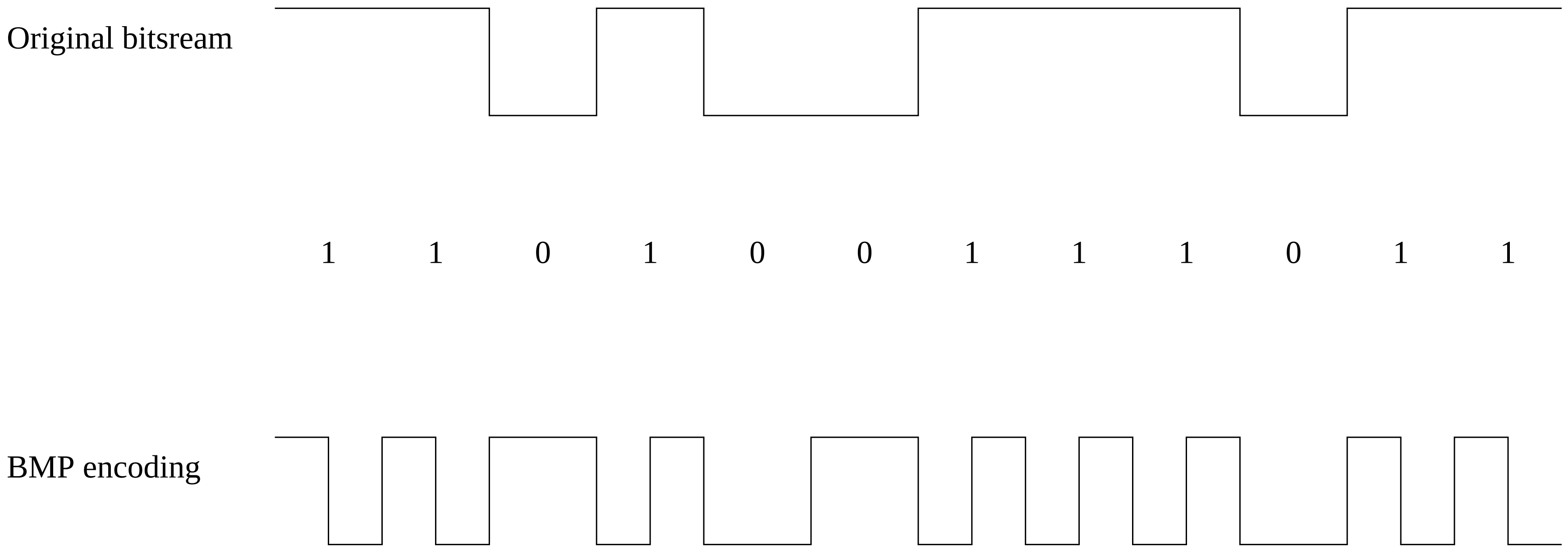
Biphase Mark is a protocol for asynchronous communication



- 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 reciever clocks run at similar rates

Biphase Mark Protocol Verification

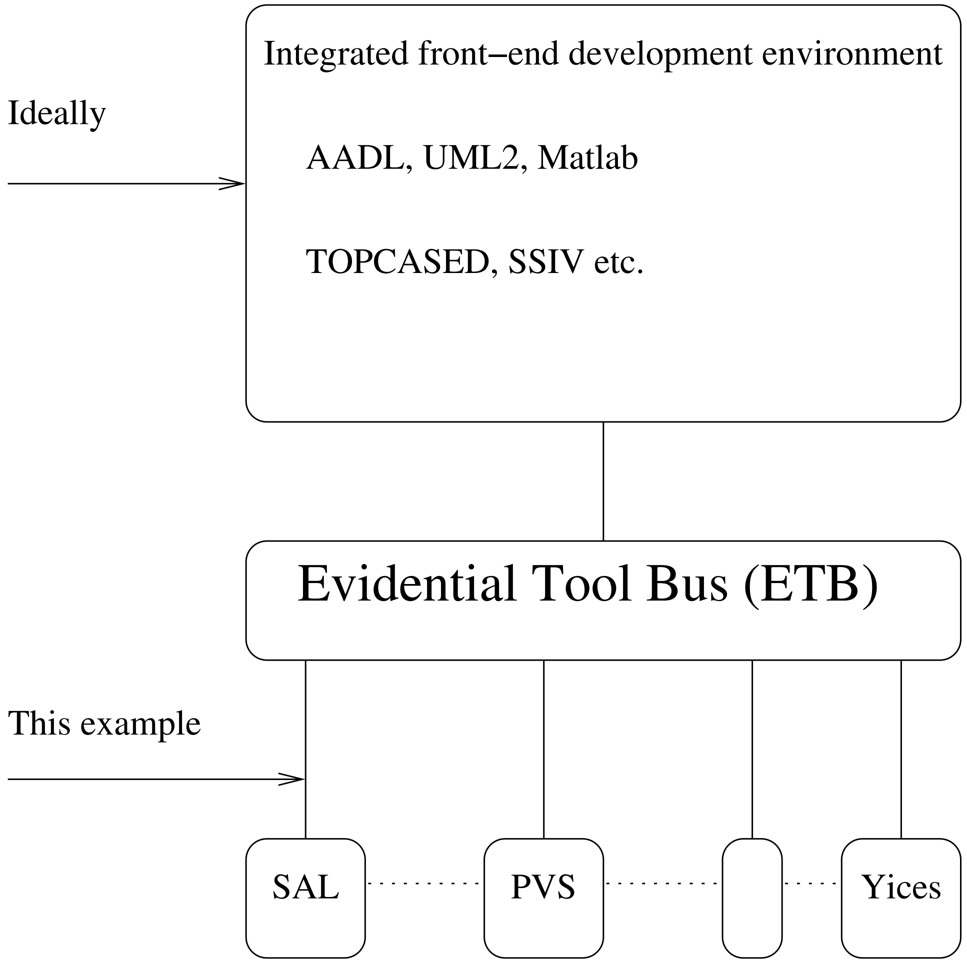
- Verified by human-guided proof in ACL2 by J Moore (1994)
- Three different verifications used PVS
  - One by Groote and Vaandrager used PVS + UPPAAL required 37 invariants, 4,000 proof steps, hours of prover time to check

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

Biphase Mark Protocol Verification (demo)

Ideally, use an integrated front end; here we look at raw model-checker input


Integrated front–end development environment
AADL, UML2, Matlab
TOPCASED, SSIV etc.
Ideally
Evidential Tool Bus (ETB)
This example
SAL
PVS
Yices

SMT Solvers as Disruptive Innovation

Disruptive to:

SAT solvers: anything a SAT solver can do, an SMT solver does better

Constraint solvers: maybe

Ordinary BMC: Infinite BMC is often faster than (finite) BMC (SMT solver on the real problem is faster than SAT on the bit-blasted representation)

CEGAR loops: SMT extends to MaxSMT and, dually, extraction of unsat cores

Traditional theorem provers: at the very least, they need an SMT solver for endgame proofs (PVS 4.0 uses Yices), but hard to rival the perfect adaptation of infinite BMC and k-induction

Summary

- Disruptive innovations are sweeping through our field
- We can both contribute to these and use them
- Contribution to disruption: SMT solvers
  - In addition to backend solvers and BMC, these can be used in automated test generation, predicate abstraction, invariant generation and verif'n, (weighted) Maxsat/ Mincore, extended static checking and extended type checking, and temporal/metric plan synthesis and verif'n
- Harness disruption: The Evidential Tool Bus
  - Lightweight decentralized open-ended architecture for integrating heterogeneous components
  - We'll use it to reconstruct Hybrid SAL, construct and integrate many methods for invariant generation
- The real targets for disruption are traditional methods for software development, validation, and certification

Thank you!

- And thanks to Bruno Dutertre, Grégoire Hamon, Leonardo de Moura, Sam Owre, Harald Rueß, Hassen Saïdi, N. Shankar, Maria Sorea, and Ashish Tiwari
- You can get our tools and papers from `http://fm.csl.sri.com`
- These slides:
  `http://www.csl.sri.com/~rushby/slides/jaist07.pdf`