ERTS 06; Toulouse January 25-7 2006

Formal Analysis
For Embedded Real-Time Systems

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

Overview

- I'm going to talk about infinite bounded model checking for real-time systems
  - It's a way of debugging and verifying models of timed systems, even in the presence of "case explosion"
    - ⋆ e.g., due to fault tolerance
- Should be new to everyone
  - And, I hope, interesting
- But I'll start by giving an introduction to model checking
  - And some demos, novel scenarios
- Which I hope will be accessible to everybody
- I'll focus on practical utility, not theory

Why Is It So Difficult. . . ?

- Why is it difficult to get systems right?
  - It's hard to think of everything up front
- Why is it difficult to get embedded systems right?
  - Have to consider environment (plant, other controllers, IMA) operating concurrently with the system
  - Possibly introducing faults
  - For fault tolerance we may then have redundant channels operating concurrently
  - So huge numbers of different behaviors
- Why is it difficult to get embedded real-time systems right?
  - Must consider all possible interleavings and durations
  - Continuous time introduces potentially infinitely many behaviors

And What Can We Do About It?

- Construct explicit models of the design and environment (including faults)
- Still hard to think of everything, but at least we have it written down
  - Others can examine it
  - If it is executable, we can do experiments
- This is what model based design (MBD) is about
- Now, suppose we could examine every behavior of the modeled design/environment interaction...

How To Examine Every Behavior?

- Reachability analysis—special case of model checking
  - Model checkers test whether a given state machine is a Kripke model for a given temporal logic formula
  - Invariants are the case: $\Box P$ or $G(P)$ or $AG(P)$
- Construct every reachable state of the system and check that desired properties (invariants) hold
  - State is an assignment of values to variables
- Simplest version: explicit state reachability analysis

Explicit State Reachability Analysis

- Imagine a simulator for some system/environment model
- Keep a set of all states visited so far, and a list of all states whose successors have not yet been calculated
  - Initialize both with the initial states
- Pick a state off the list and calculate all its successors
  - i.e., run all possible one-step simulations from that state

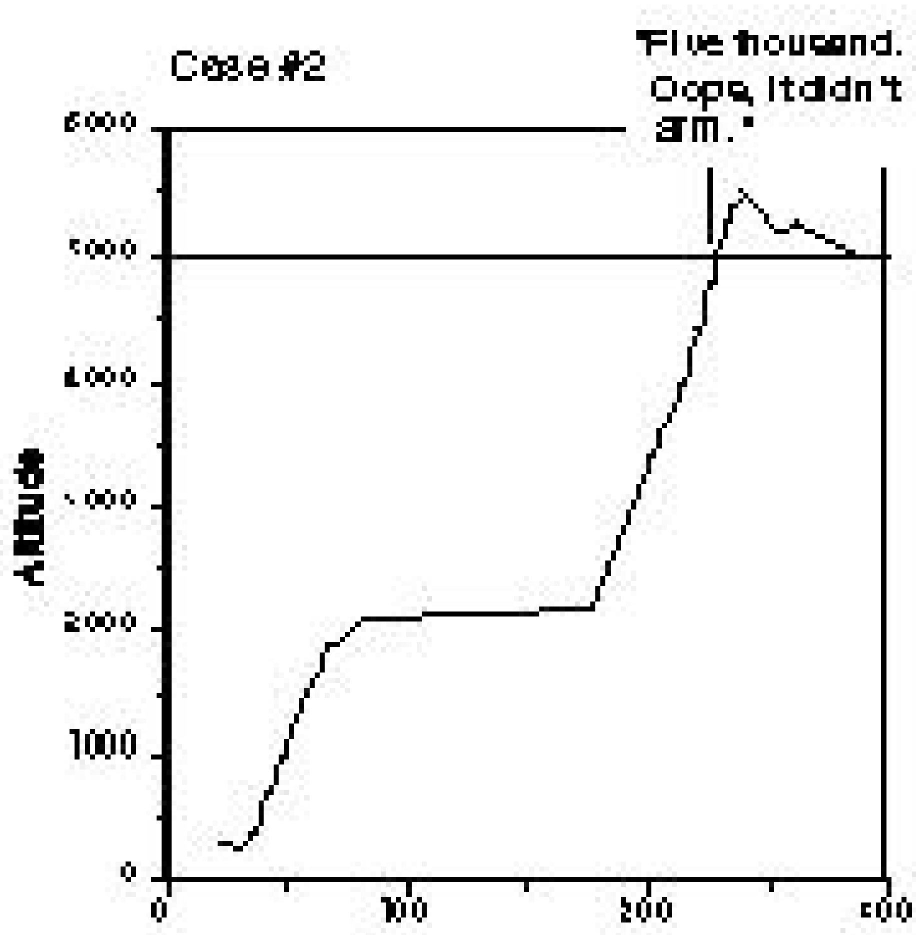
  Throw away those seen before
- Add new ones to the set and the list
- Check each new state for the desired properties
- Iterate to termination, or some state fails a property
  - Or run out of memory, time, patience
- On failure, counterexample (backtrace) manifests problem

Explicit State Reachability Analysis: Example

- Not limited to modeling electronic systems
- Here, we'll model the pitch mode transitions of the MD88 autopilot
- And those of a mental model
  - Suggested by the training manual
- And check the property that these always agree on whether capture mode is active
- Demo: `sal-esmc -v 3 md88 no_surprise`
- This scenario was previously observed by NASA in a flight simulator: a famous automation surprise
  - "Whoops it didn't arm!"

Observed Automation Surprise: An Altitude Bust

- Plane passed through 5,000 feet at vertical velocity of 4,000 fpm
- "Oops: It didn't arm"
- Captain took manual control, halted climb at 5,500 with the "*altitude—altitude*" voice warning sounding repeatedly



From Explicit to Symbolic Model Checking

- Explicit state model checkers run out of steam around 10-100 million reachable states
- But that's only around 25 state bits
- Can often represent states more compactly using symbolic representation
- E.g., the infinite set of states $\{(0,1),(0,2),(0,3),\ldots(1,2),(1,3),\ldots(2,3),\ldots\}$ can be symbolically represented as the finite expression $\{(x,y)\,|\,x<y\}$
- Symbolic model checkers use such symbolic representations

Symbolic Model Checking

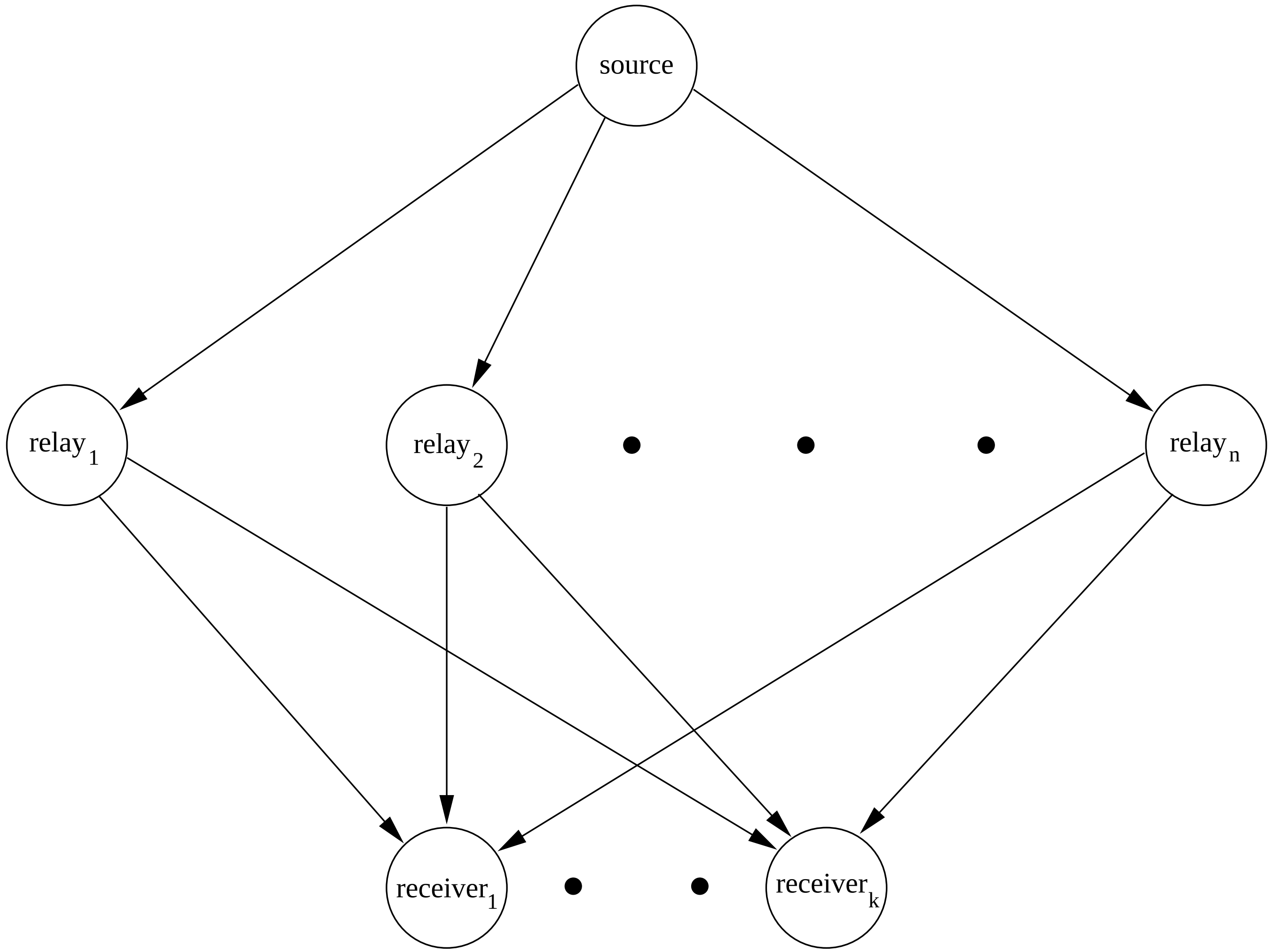
- Compile the model to a Boolean transition relation T
  - i.e., a circuit
- Initialize the Boolean representation of the stateset S to the initial states I
- Repeatedly apply T to S until a fixpoint
  - $S' = S \cup \{t \,|\, \exists s \in S : T(s,t)\}$
  - Final S is a formula representing all the reachable states
- Check the property against final S
- Mechanized efficiently using BDDs
  - Reduced ordered Binary Decision Diagrams

  Commodity software, honed by competition (CUDD)

Symbolic Model Checking: Example

- We'll model the OM(1) algorithm for source congruence (aka. Byzantine Agreement, interactive consistency)
- Needed whenever a single source (e.g., sensor) is distributed to multiple channels (e.g., redundancy for fault tolerance)
  - Faulty source (e.g., sending weak voltages) could otherwise drive the channels apart
- Solution is to pass through n intermediate relays in parallel and vote the results

OM(1)



Can tolerate certain numbers and kinds of faults:

use model checking to explore which ones

From Symbolic to Bounded Model Checking

- Demo: `sal-smc -v 3 om1 agreement`
- With 3 relays, 10,749,517,287 reachable states
- With 4 relays, 66,708,834,289,920 reachable states
- With 5 relays, run out of patience finding counterexample to validity property
- Modern SMC can handle 600 state bits before special tricks are needed, seldom get beyond 1,000 state bits
- Bounded model checkers are specialized to finding counterexamples
- Sometimes can handle bigger problems than SMC

Bounded Model Checking

- Is there a counterexample to P in k steps or less?
- Does there exist assignments to states $s_0, \ldots, s_k$ such that

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

- Given a Boolean encoding of I, T, and P (i.e., circuit), this is a propositional satisfiability (SAT) problem
- SAT is the quintessential NP-Complete problem
- But current SAT solvers are amazingly fast
- Commodity software, honed by competition (MiniSAT, Siege, zChaff, Berkmin)
- BMC uses same representation as SMC, different backend
- Demo: `sal-bmc -v 3 om1 validity -d 3`

Test Generation

- Observe that counterexample to: "control cannot reach this point" is a structural test case
- So BMC can be used for automated test generation
- Actually, a customized combination of SMC and BMC works best
  - Use SMC to reach first control point, then use BMC to extend to further control points
  - Get long tests that probe deep into the system
  - Can add test purposes that constrain the kinds of tests generated
    - ⋆ e.g., Change the gear input by 1 at every step
  - Easily built because checkers are scriptable (in Scheme)

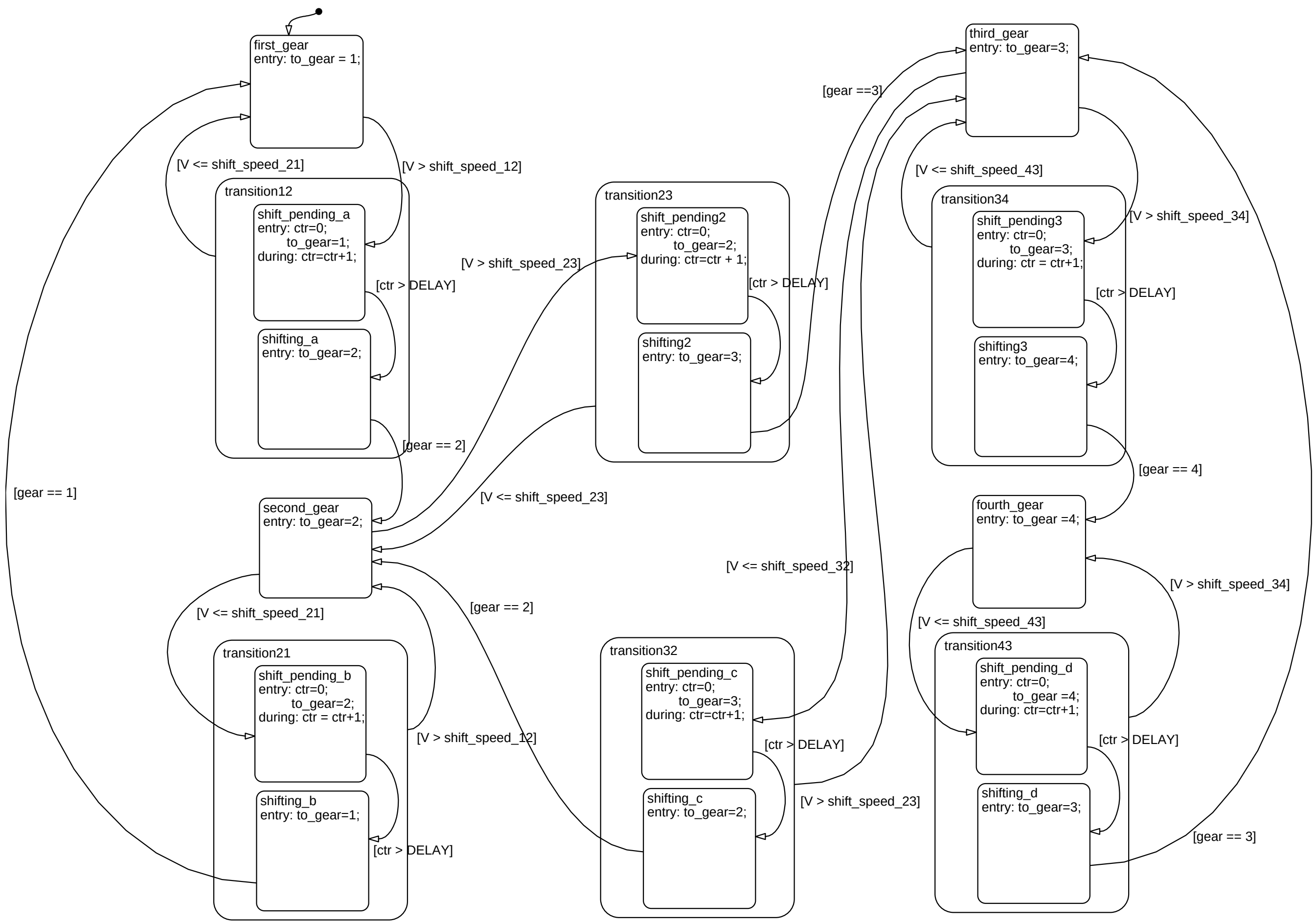
Core Of The SAL-ATG Test Generation Script

```
(define (extend-search module goal-list
         path scan prune innerslice start step stop)
   (let ((new-goal-list (if prune (goal-reduce scan goal-list path)
                             (minimal-goal-reduce scan goal-list path))))
    (cond ((null? new-goal-list) (cons '() path))
          ((> start stop) (cons new-goal-list path))
          (else
           (let* ((goal (list->goal new-goal-list module))
                  (mod (if innerslice
                           (sal-module/slice-for module goal) module))
                  (new-path
                   (let loop ((depth start))
                        (cond ((> depth stop) '())
                              ((sal-bmc/extend-path
                                   path mod goal depth 'ics))
                              (else (loop (+ depth step)))))))
             (if (pair? new-path)
                 (extend-search mod new-goal-list new-path scan
                                    prune innerslice start step stop)
               (cons new-goal-list path)))))))
```

Outer Loop Of The SAL-ATG Test Generation Script

```
(define (iterative-search module goal-list
          scan prune slice innerslice bmcinit start step stop)
  (let* ((goal (list->goal goal-list module))
         (mod (if slice (sal-module/slice-for module goal) module))
         (path (if bmcinit
                   (sal-bmc/find-path-from-initial-state
                       mod goal bmcinit 'ics)
                 (sal-smc/find-path-from-initial-state mod goal))))
    (if path
        (extend-search mod goal-list path scan prune
            innerslice start step stop)
        #f)))
```

Example: Shift Scheduler in StateFlow



Demo: `sal-atg -v 3 trans_ga monitored_system trans_ga_goals.scm -id 15 -ed 7 --testpurpose`

Verification with BMC

- BMC was originally developed for refutation (bug finding)
- But can be used for verification via k-induction
- 1-induction; 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 (cf. strong induction):

  Basis: No counterexample of length k or less

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

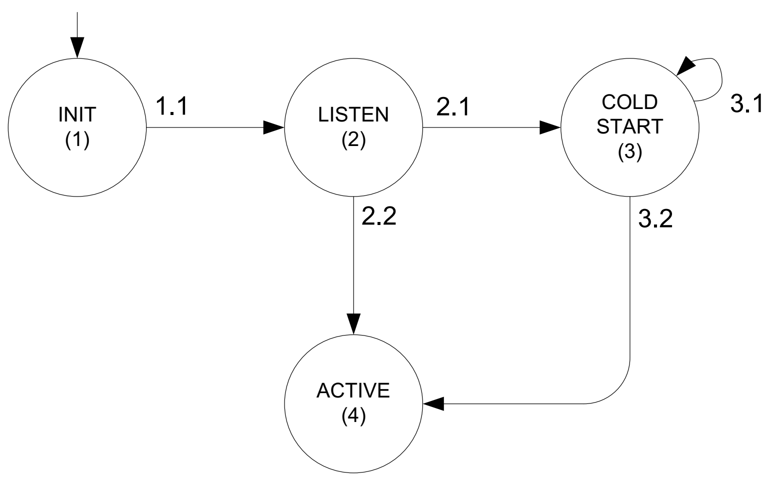
  These are close relatives of the BMC formulas
- Induction for $k = 2, 3, 4 \ldots$ may succeed where $k = 1$ does not
- Demo: `sal-bmc -v 3 om1 agreement -d 4 -i`

Timed Systems

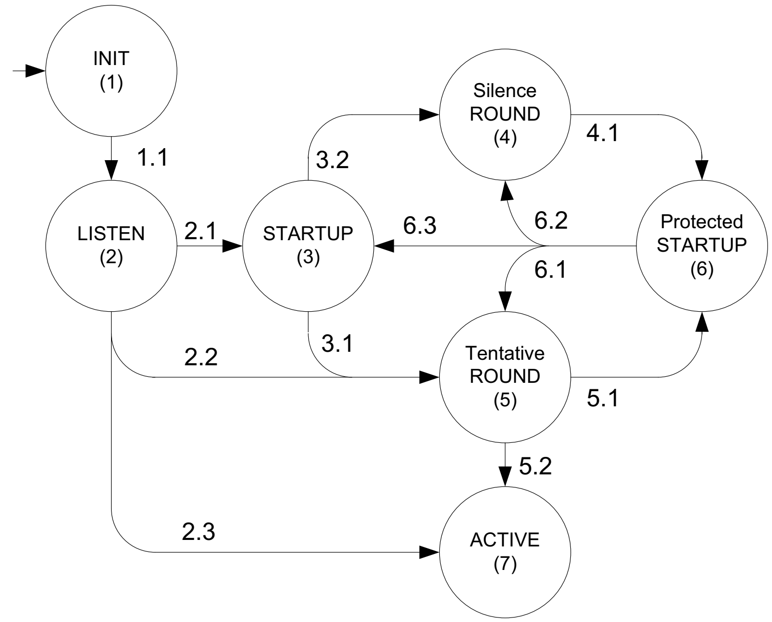
- Simplest notion of time simply counts events
- Example: TTA startup
- TTA (Time Triggered Architecture) is an IMA bus
  - Used e.g., in FADECs for F16 and Aeromachi trainer
- May need to restart in flight
  - e.g., following massive HIRF event
- Must happen in bounded time, in presence of faults
- During startup controllers are operating asynchronously
  - After period of silence, send startup signal
  - May collide, so backoff
  - Show number of collisions is bounded
    - ⋆ And find the bound

Startup in TTA: State Machines

Nodes



Hubs



- There are two hubs, n nodes, each component can wake up at a slightly different time
- Also different numbers and kinds of faults my be present

Analyzing TTA Startup by Model Checking

- Have "dials" on value of n and intensity of faults
- Allows us to vary the difficulty of the model checking problem from a few minutes (during development and exploration) to overnight (for verification)
- Biggest case are big!
- E.g., 259,220,300,300,290 states (10^{15}) with 5 nodes
- Able to find sharp bound on worst case startup delay

Clocked Systems

- Next kind of timed system is one with a discrete clock
- In modeling, add the clock as a component
  - All it does it output ticks
- Clock ticks are counted just like the events in the previous example
- Fine-grain clocks generate large statespace and will overwhelm the model checker
  - Can be improved using calendars and timeouts (see later)

Continuous (i.e., Real) Time

- Infinitely many instants between any pair of events
- Hence, any model that includes a representation of continuous time is infinite state
- Ordinary model checking assumes finite state
- There are specialized model checkers for timed automata
  - Represent time constraints by sets of polyhedra
  - Efficient methods for representing and operating on these
  - But these must be combined with representations for the discrete components of the state

  Hence timed automata can get overwhelmed by the "case explosion" when fault tolerance is added to real time

Infinite Bounded Model Checking

- Recall that bounded model checking: seeks counterexample to property p in k steps or fewer
- Requires assignments to states $s_0, \ldots, s_k$ such that

  $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg(P(s_1) \wedge \cdots \wedge P(s_k))$
- Previously, we used a Boolean encoding of I, T, and P
- Suppose, instead, we used Booleans, plus terms from decidable theories, such as linear real arithmetic, integer arithmetic, arrays, etc.
- Instead of a Boolean SAT problem we have an SMT problem
  - Satisfiability Modulo Theories
- Result is a bounded model checker for infinite state systems, aka. an infinite bounded model checker

SMT

- Individual decision procedures decide conjunctions of formulas in their decided theories
- Combinations of decision procedures (using, e.g., Nelson-Oppen or Shostak methods) decide conjunctions over the combined theories (e.g., arithmetic plus arrays)
- 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
  - $(A \vee B) \wedge (C \vee D) \wedge E$
- Get a solution from a SAT solver (if none, we are done)
  - e.g., A, D, E

Lemmas On Demand

- Restore the interpretation of variables and send the conjunction to the core decision procedure
  - e.g., $x \leq y \wedge y \leq x \wedge x \neq y$
- 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)
  - $A \wedge D \supset \neg E$
- Iterate to termination (e.g., B, D, E: $y = 5, y < x$: $y = 5, x = 6$)
- We call this "lemmas on demand" or "lazy theorem proving"
- it works really well: our system is called ICS

SMT Solvers

- SMT solvers are being honed by 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
  - Arrays
- Yices and Simplics (prototypes for next ICS) won all hard divisions and came second in all the easy ones

Verification by Infinite Bounded Model Checking

- Infinite BMC extends from refutation (counterexamples) to verification using k-induction, just like ordinary BMC
- SMT solvers provide the horsepower
- Even though k-induction is much stronger than 1-induction, may still need to strengthen the invariant
  - Disjunctive invariants work well in these examples

Real Time Analysis by Infinite Bounded Model Checking

- We'll have a model component representing time
- Problem is: how does this component advance time?
- Use timeout automata:
  - Has an array with an entry for each (other) component indicating time when that component will next do something (its timeout)
  - When all other components are blocked, timeout automaton advances time to earliest timeout
- Other components
  - Make a move when time equals their timeout
  - Then block and set timeout to time of next move
  - Note timeouts can be nondeterministic (i.e., intervals)
- Similar to the way discrete event simulation systems work

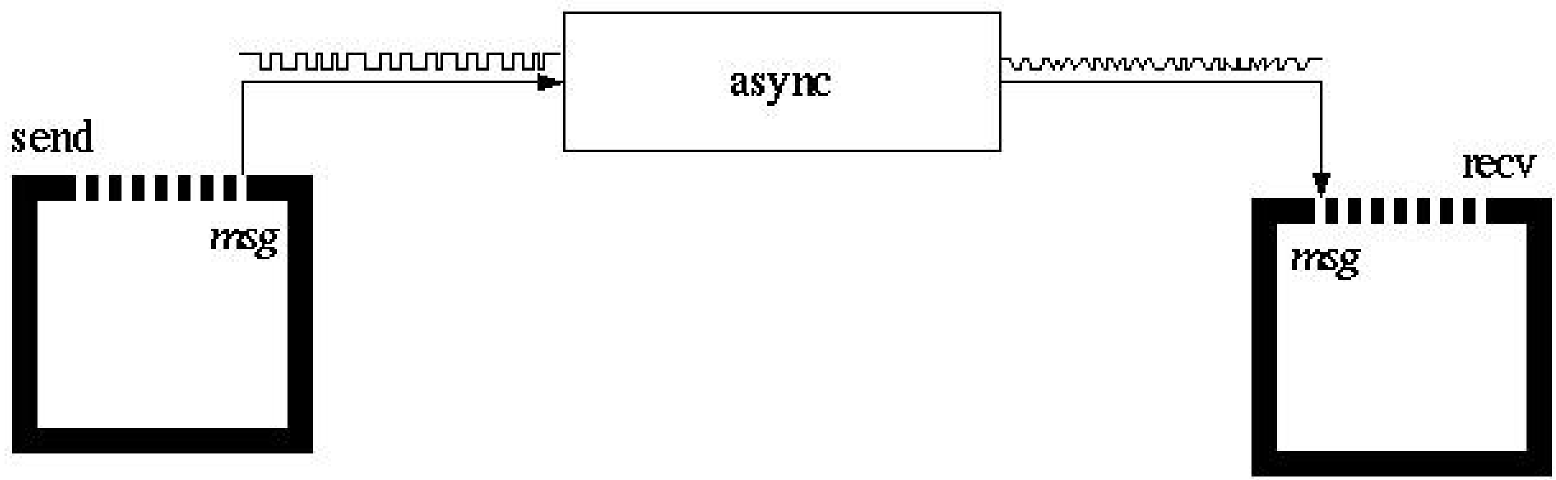
Real Time Analysis with Infinite Bounded Model Checking

- Timeout automata are fast on standard benchmarks
  - e.g., Fischer's real-time mutual exclusion with 43 processes
- They were developed by Dutertre and Sorea
- Who applied them to real-time version of TTA startup
- Simplified and optimized by Pike and Brown
  - SPIDER (IMA bus) reintegration protocol
    - ⋆ Fault tolerant and real time
  - Several data communication protocols
- Examples using events as well as clocks need more complex clock component
  - Calendar automata (Dutertre and Sorea)

Performance of InfBMC for Real Time

- Biphase Mark Protocol is an algorithm for asynchronous communication
  - Clocks at either end may be skewed and have different rates, 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
- 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



Biphase Mark Protocol

- 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
  - Demo:

    `sal-inf-bmc -v 3 -d 5 -i -l l0 -l l1 -l l2 biphase t0`
- Adapted verification to 8-N-1 protocol (used in UARTs)
  - Additional lemma proved with 13-induction
  - Theorem proved with 3-induction (7 disjuncts)
  - Revealed a bug in published application note

Prospects: Near Term Topics

- k-induction requires lemmas and strengthened invariants
  - Should investigate direct construction of reachable states
  - Like timed automata model checkers
- Modeling notations of model checkers differ from those of MBD systems (Simulink/Stateflow and Esterel/SCADE)
  - Need semantics for MBD notations (e.g., Caspi, Hamon)
  - Can then translate from MBD to InfBMC
- Test generation for avionics code needs SMT for undecidable theories (trigonometric functions, nonlinear arithmetic)
  - But can tolerate unsoundness (Xia, Di Vito, Muñoz)

Prospects: Medium Term

- Possible application of SMT solvers to hybrid systems (state machines plus differential equations)
  - But automated abstractions do very well (Tiwari)
  - Uses fast decision procedures for real closed fields
  - Should examine this approach for timed systems
- And should look at test generation for timed and hybrid systems
  - May not have full control of the plant
  - So tester is a program, not a sequence of inputs
  - Need to extend from model checking to controller synthesis
    - ⋆ Scheduling could use similar techniques

Larger Prospects

- The raw power of SMT solvers could revolutionize many formal analysis tasks
- Especially when combined with other recent advances: predicate abstraction, counterexample-guided abstraction refinement (CEGAR), Craig interpolants, static analysis methods, etc.
- Could soon be feasible to build very effective extended static checkers (cf. ESC Java), software model checkers (cf. Blast), and automated verifiers
- Need a way to combine analyses from many sources to yield larger ones
  - Cf. "The Evidential Tool Bus"

Even Larger Prospects

- The raw power of SMT solvers could revolutionize some AI tasks
  - Anything SAT can do, SMT does better
  - SMT extends to MaxSMT as SAT extends to MaxSAT
    - Given unsatisfiable set of weighted formulas, find satisfiable subset of maximum weight
    - Used in model based diagnosis, integrating learners
- And constraint solving
  - Can find SMT assignment that maximizes any given arithmetic expression
    - Used in plan generation
- We are just starting work on assurance for autonomous manned spacecraft and hope to explore these topics

To Learn More

- Our systems, PVS, SAL, ICS and our papers are all available from http://fm.csl.sri.com
- Thanks to Bruno Dutertre, Grégoire Hamon, Leonardo de Moura, Sam Owre, Harald Rueß, Hassen Saïdi, N. Shankar, and Maria Sorea