HSCC invited talk, Wednesday 29 March 2006

# Hybrid Systems
# . . . And Everything Else

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

## Overview

- Some of the computer science (mostly fault tolerance)
- That supports the hybrid systems at the core many embedded systems
- Look at each step of the sense, compute, act cycle
- And some systems issues: self stabilization, IMA, human interaction
- Formal analysis: SMT solvers
- Tying analyses together: an Evidential Tool Bus

## **Sense: Communicating a Single Sensor Sample—1**

Traditional Approach

**How good is it?** unknown

**Is it valid?** maybe not

- Ariane 501: complex scenario, leading to
- Diagnostic output interpreted as flight data, then
- Full nozzle deflections of solid boosters and loss of vehicle

**Is it stuck?** zero on read

**How old is it?** timestamps

## Sense: Communicating a Single Sensor Sample—2

**Intelligent sensor** communicates an interval in which the true value is sure to lie (for a nonfaulty sensor)

**How good is it?** width of interval

**Is it valid?** infinite interval, or separate status

**Is it stuck?** handled as below

**How old is it?** sent with use by time

**Embellishment** interval is a function of time

## Sense: Fusing Multiple Sensor Samples—1

Traditional Approach (e.g., with 3 samples)

**Eliminating faulty samples:**

Reject if not within 15% of the others

**Fusing for a single value:**

Mid-value select when 3, average when 2

**Problems:** thumps and bad values

## X29

- Three sources of air data: a nose probe and two side probes
- Selection algorithm used the data from the nose probe, provided it was within some threshold of the data from both side probes
- The threshold was large to accommodate position errors in certain flight modes
- If the nose probe failed to zero at low speed, it would still be within the threshold of correct readings, causing the aircraft to become unstable and "depart"
- Found in simulation
- 162 flights had been at risk

## Sense: Fusing Multiple Sensor Samples—1 (ctd.)

- Recent methods use more complex selection algorithms
- Take the dynamics into account
- Hence, they are hybrid systems
- Here's one specified in Simulink (page 5)

## Sense: Fusing Multiple Sensor Samples—2

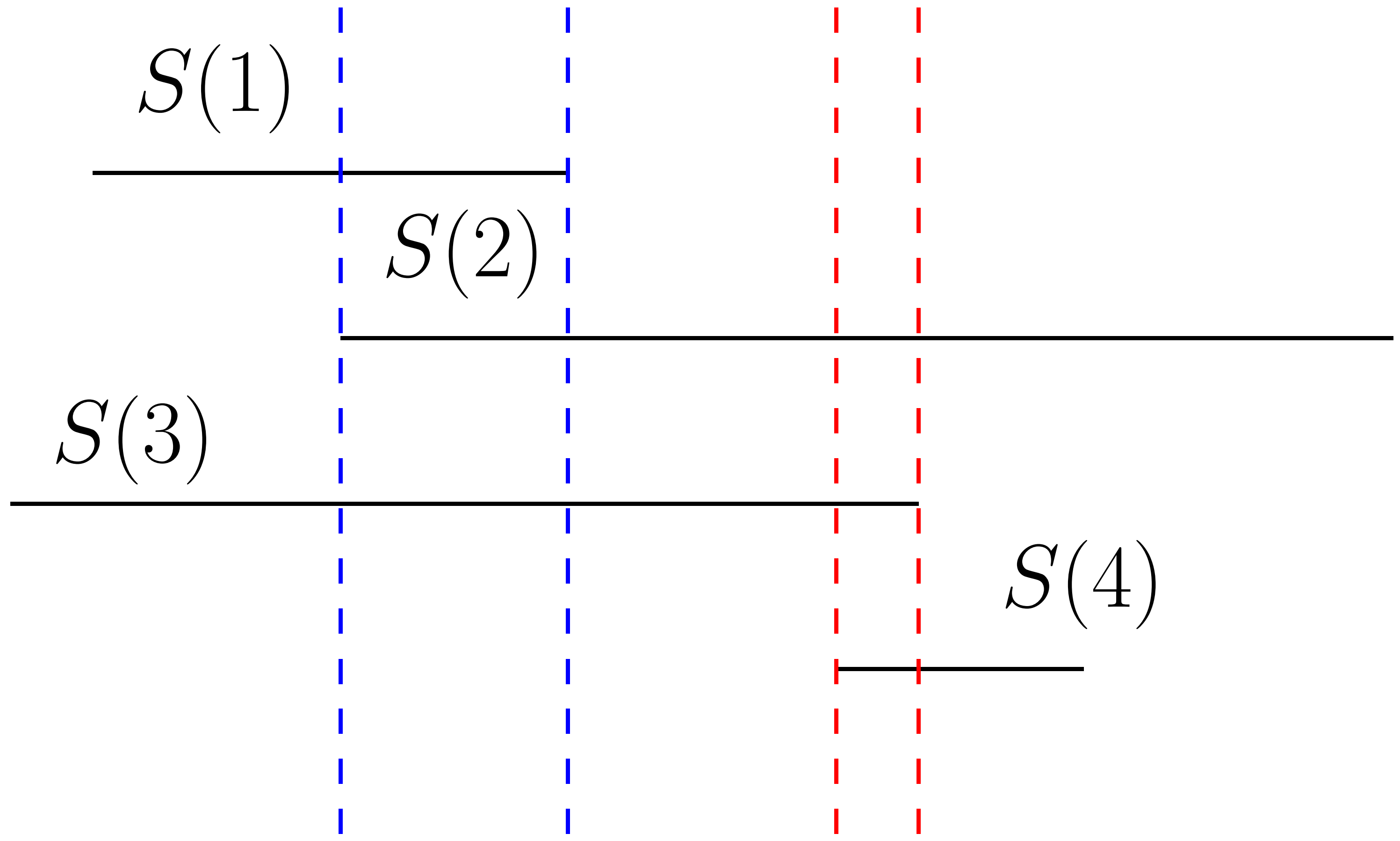
**Interval approach:** true value must be in overlap of nonfaulty intervals

**Fusing for a single value** to tolerate $f$ faults in $n$, choose interval that contains all overlaps of $n - f$;

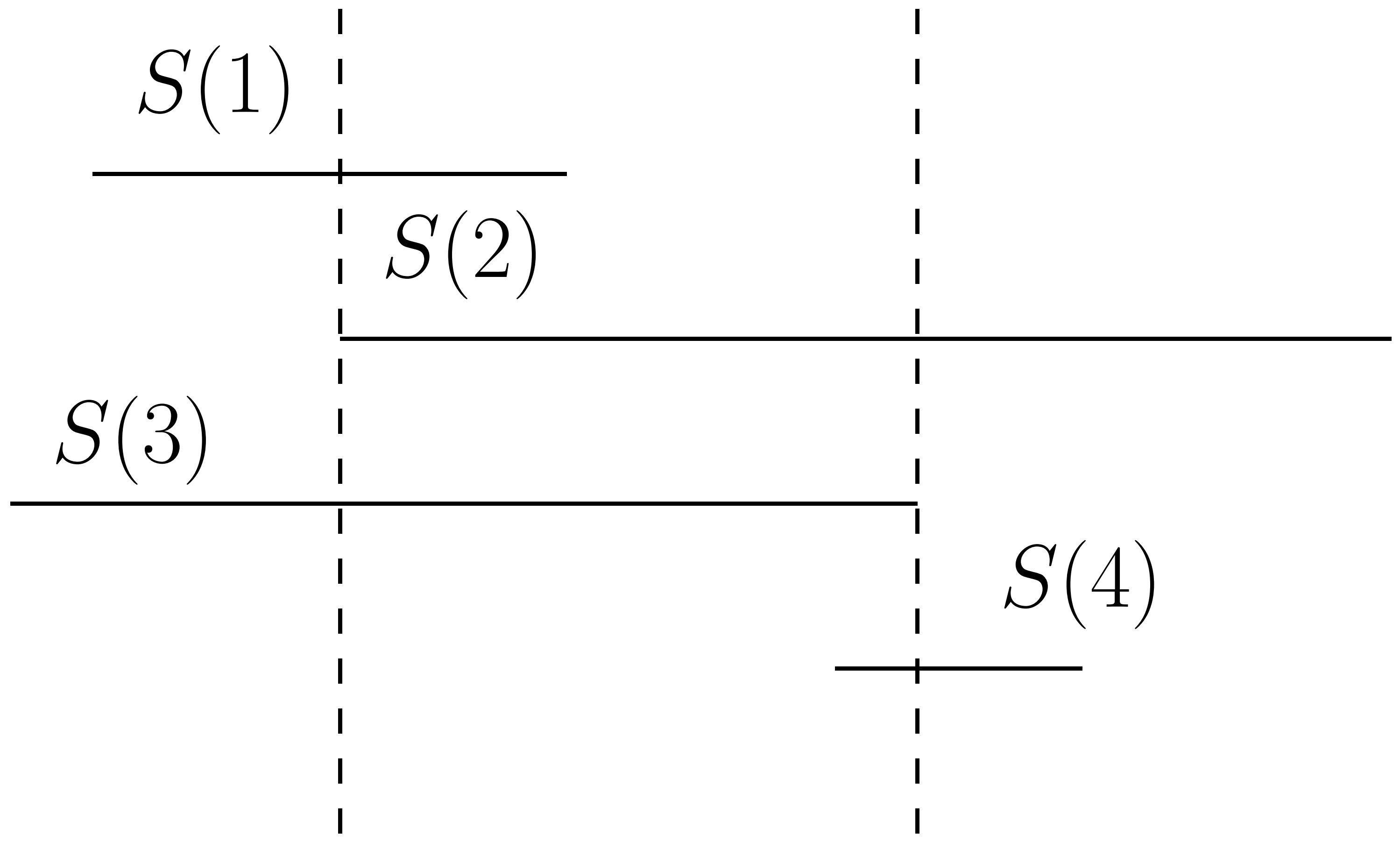
i.e., from least value contained in $n - f$ intervals to largest value contained in $n - f$ (Marzullo)

**Eliminating faulty samples:** separate problem, not needed for fusing, but any sample disjoint from the fused interval must be faulty

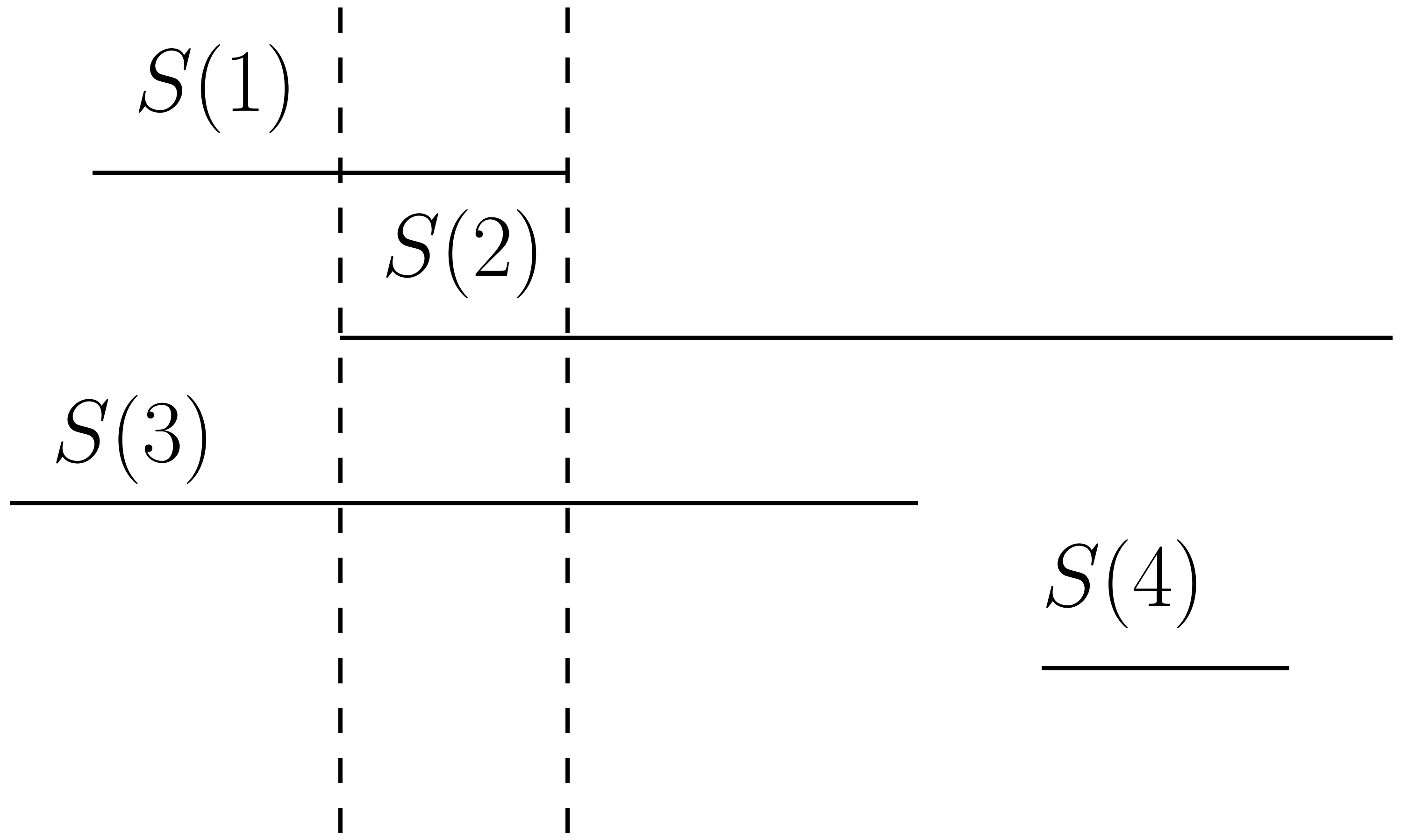
## True Value In Overlap Of Nonfaulty Intervals



## Marzullo’s Fusion Interval



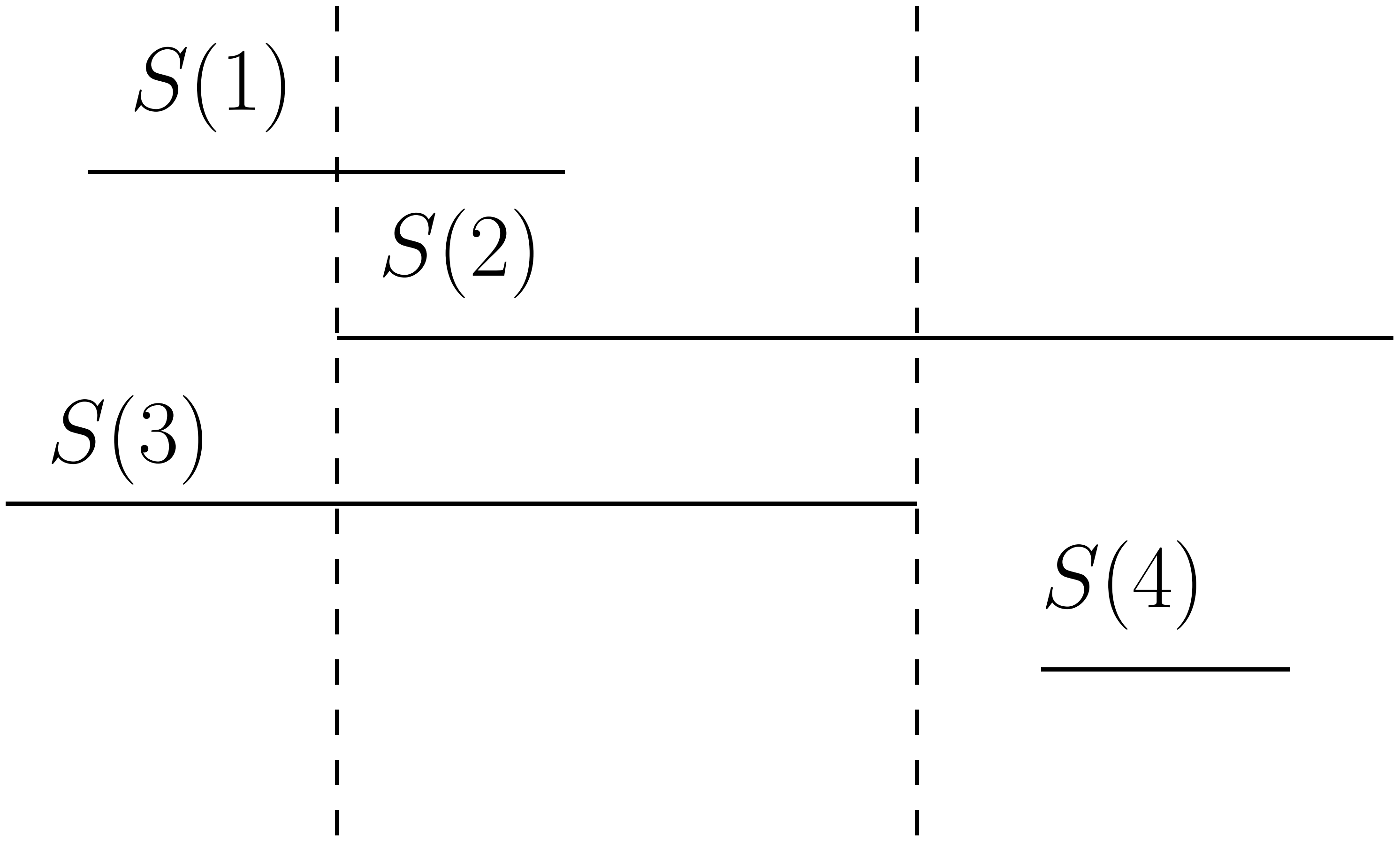
## Marzullo's Fusion Interval: Fails Lipschitz Condition



## Schmid's Fusion Interval

- Choose interval from $f+1$'st largest lower bound to $f+1$'st smallest upper bound
- Optimal among selections that satisfy Lipschitz Condition

## Schmid’s Fusion Interval



## **Compute: Redundancy For Fault Tolerance**

Several approaches

**Self-checking pairs:** later

**N-modular redundancy:** unsynchronized or synchronized?

**Unsynchronized:** channels sample sensors independently, compute independently; outputs can be selected (like sensors), voted, or averaged

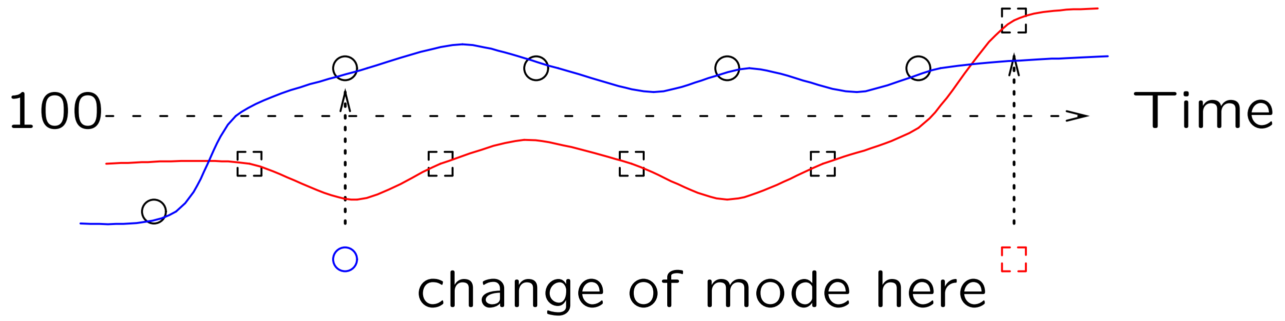
- Intuitively maximizes diversity, independence
- Homespun solution
- A mass of problems in practice

## Problems with Unsynchronized Designs

- Channel outputs depend on time integrated values (e.g., velocity, position)
    - Accumulated errors are compounded by clock drift
    - Must exchange and vote integrator values
    - Requires synchronization in the applications code
- In general, redundancy management pervades applications code (as much as 70% of the code)

## Problems with Unsynchronized Designs (ctd.)

- Output selection can induce large transients
  - Averaging functions dragged along by faulty values
  - Exclusion on fault detection causes drastic change
- Mode switches can cause channel divergence
  - IF $x > 100$ THEN . . . ELSE . . .



  - Output very sensitive to sample when near decision point
- Have to modify control laws to ramp changes in and out smoothly, or use ad hoc synchronization and voting

## Historical Experience of DFCS (early 1980s)

- Advanced Fighter Technology Integration (AFTI) F16
- Digital Flight Control System (DFCS) to investigate "decoupled" control modes
- Triplex DFCS to provide two-fail operative design
- Analog backup
- Digital computers not synchronized
- "General Dynamics believed synchronization would introduce a single-point failure caused by EMI and lightning effects"

## AFTI F16 Flight Test, Flight 36

- Control law problem led to “departure” of three seconds duration
- Sideslip exceeded $20^\circ$, normal acceleration exceeded $-4$g, then $+7$g, angle of attack went to $-10^\circ$, then $+20^\circ$, aircraft rolled $360^\circ$, vertical tail exceeded design load, failure indications from canard hydraulics, and air data sensor
- Side air data probe blanked by canard at high AOA
- Wide threshold passed error, different channels took different paths through control laws
- Analysis showed this would cause complete failure of DFCS and reversion to analog backup for several areas of flight envelope

## AFTI F16 Flight Test, Flight 44

- Unsynchronized operation, skew, and sensor noise led each channel to declare the others failed
- Simultaneous failure of two channels not anticipated
  So analog backup not selected
- Aircraft flown home on a single digital channel
  (not designed for this)
- No hardware failures had occurred

## Other AFTI F16 Flight Tests

- Repeated channel failure indication in flight was traced to roll-axis software switch
- Sensor noise and unsynchronized operation caused one channel to take a different path through the control laws
- Decided to vote the software switch
- Extensive simulation and testing performed
- Next flight, same problem still there
- Found that although switch value was voted, the unvoted value was used

## Analysis: Dale Mackall, NASA Engineer
## AFTI F16 Flight Test

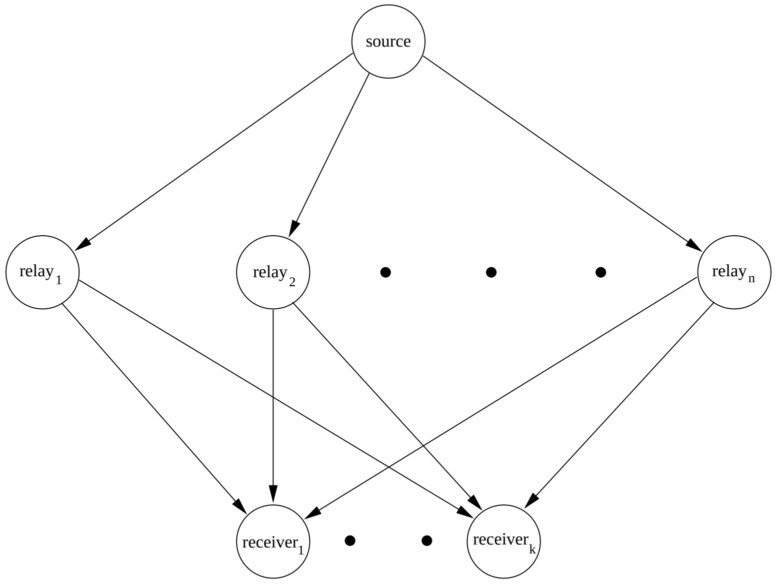
- Nearly all failure indications were not due to actual hardware failures, but to design oversights concerning unsynchronized computer operation
- Failures due to lack of understanding of interactions among
    - Air data system
    - Redundancy management software
    - Flight control laws (decision points, thumps, ramp-in/out)

## Synchronized Fault-Tolerant Systems

- Synchronized systems can use exact-match voting for fault-masking and transient recovery—potentially simpler and more predictable
- It's easier to maintain order than to establish order (Kopetz)
  - Synchronized designs solve the hard problems once
  - Unsynchronized designs must solve them on every frame
- Need fault-tolerant clock synchronization
- And fault-tolerant distribution of sensor values so that each channel works on the same data: interactive consistency (aka. source congruence, Byzantine agreement)
- Both these need to deal with asymmetric or Byzantine faults

## Interactive Consistency

- Needed whenever a single source (e.g., sensor) is distributed to multiple channels (e.g., redundancy for fault tolerance)
  - Faulty source could otherwise drive the channels apart
- A solution is to pass through $n$ intermediate relays in parallel and vote the results (OM(1) algorithm)



Can tolerate certain numbers and kinds of faults

## SOS Interpretation of Byzantine Faults

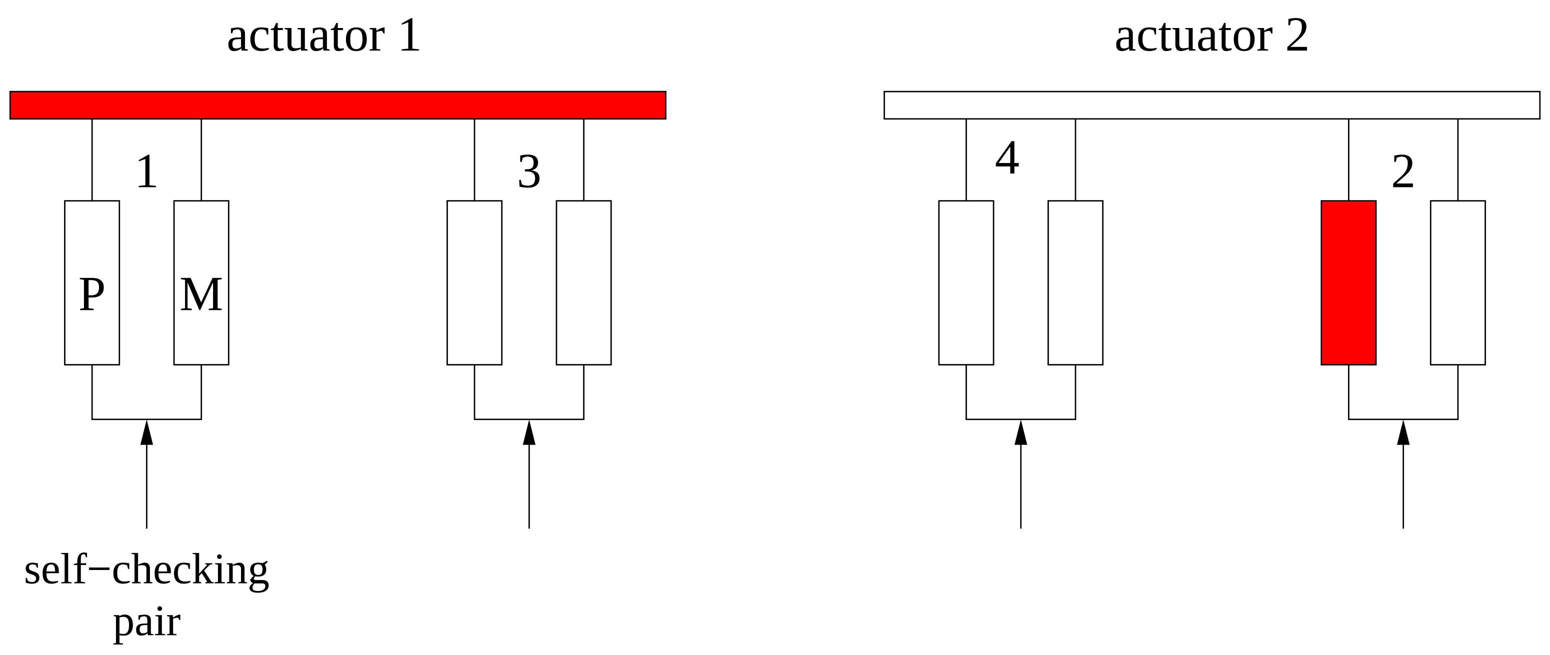
- The “loyal” and “traitorous” Byzantine Generals metaphor is unfortunate
  - Also academic focus on asymptotic issues rather than maximum fault tolerance from given resources
- Leads most homespun designers to reject the problem
  - Also, $10^{-9}$ per hour is beyond casual human experience
  - Actual frequency of rare faults is underestimated
- Slightly Out of Specification (SOS) faults can exhibit Byzantine behavior
  - Weak voltages (digital 1/2)
    - ⋆ One receiver may interpret 2.5 volts as 0, another as 1
  - Edges of clock regions
    - ⋆ One receiver may get the message, another may not

## A Real SOS Fault

- Massively redundant aircraft system
- Theoretically enough redundancy to withstand 2 Byzantine faults
- But homespun design did not consider such possibility
- Several failures in 2 out of 3 “independent” units
- Entire fleet within days of being grounded
- Adequate fix developed by engineer who had designed a Byzantine-resilient system for same aircraft

## Act: Dealing with Actuator Faults

- One approach, based on self-checking pairs does not attempt to distinguish computer from actuator faults
- Must tolerate one actuator fault and one computer fault simultaneously



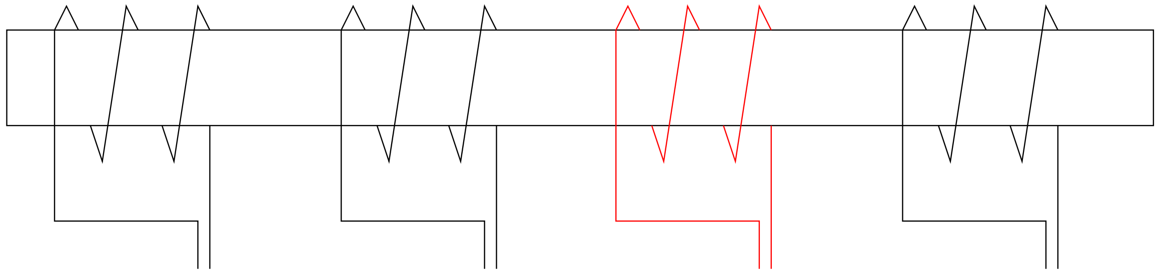
- Can take up to four frames to recover control

## Act: Consequences of Slow Recovery

- Use large, slow moving ailerons rather than small, fast ones
  - Hybrid systems verification question: is this right?
- As a result, wing is structurally inferior
- Holds less fuel
- And plane has inferior flying qualities
- All from a choice about how to do fault tolerance

## Act: Physical Averaging At The Actuators

- Alternative uses averaging at the actuators
  - E.g., multiple coils on a single solenoid



  - Or multiple pistons in a single hydraulic pot
- Hybrid systems verification question: how well can this work?

## Malaysian Air 777 Incident Near Perth

Crew got instrument indications that the aircraft was approaching the overspeed limit and the stall speed limit simultaneously

The aircraft pitched up and climbed to approximately FL410 and the indicated airspeed decreased from 270 kts. to 158 kts. The stall warning and stick shaker devices also activated

The captain disconnected the autopilot and lowered the nose of the aircraft. The autothrottle commanded an increase in thrust which the captain countered by manually moving the thrust levers to the idle position. The aircraft pitched up again and climbed 2,000 ft.

Both left and right autopilots caused the aircraft to bank and the nose to pitch down

## Malaysian Air 777 Incident Near Perth (ctd.)

- Immediate cause was abrupt and persistent offsets in Air Data Inertial Reference Unit (ADIRU) acceleration outputs
- Plane dispatched with known fault in ADIRU
- But ADIRU is massively redundant and can tolerate 2 faults
- Let's speculate. . .

## Self Stabilization

- Ability of a distributed system to converge to a good state from arbitrary initial state
- Needed in transient recovery: HIRF or other upset makes processor do bad things, then it recovers
    - But state data is damaged
    - Processor may not know this, so cannot execute special recovery process
- Self stabilization must coexist with normal activity
    - Detectors and correctors (Arora and Kulkarni)
- Must work in presence of some permanent faults and ongoing fault tolerance
    - Somewhat different to academic assumptions

## Integrated Modular Avionics (IMA)

- Multiple functions sharing a common computer resource
  - Maybe of different criticality levels
- Partitioning must be assured to about $10^{-10}$ per hour
- Challenging issues in RTOS verification
  - MILS is similar, for security
- And even more in the buses that connect them
  - SAFEbus (777 AIMS)
  - TTA (FADECs for F16 and Aeromachi trainer)
  - Spider (NASA)
  - FlexRay (cars, probably not for safety-critical functions)

These have to solve all the problems considered previously
(plus wait-free, lock-free atomic registers)

## Human Interaction

- Dominant cause of incidents and accidents is human error
- Often provoked by bad design
- Humans construct a mental model of the automation
    - Maybe a qualitative hybrid system
- Actual system behavior should be consistent with a plausible mental model (what's in the operator's manual)
- So build an explicit mental model and look for divergences from actual system model
- Discrete models find known automation surprises in
    - MD88 pitch modes
    - A320 speed protection
    - 737 pitch modes

Claire Tomlin and students analyzed hybrid systems models

## Formal Analysis and Verification

- With colleagues, I've formally verified variants of most of these topics and other related issues
- Mostly by interactive theorem proving with PVS and its predecessors
- Some by model checking in SAL
  - Imposes some modeling limitations: strictly a Byzantine fault is one for which you make no assumptions (except it does not violate FCU boundaries)
  - But for model checking you must specify some behavior
- There's now potential to do better
  - The expressiveness of theorem proving
  - With the automation of model checking
- Thanks to SMT solvers

## SMT Solvers

- SMT stands for Satisfiability Modulo Theories
- SMT solvers generalize SAT solving by adding the ability to handle arithmetic and other decidable theories
- SAT solvers are used for
  - Backends to interactive theorem provers
  - Bounded model checking, and
  - AI planning,

  among other things
- Anything a SAT solver can do, an SMT solver can do better

## 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
    - 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

## 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 they work 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
    $f \ \textbf{with}\ [(x) := y](z) \overset{\text{def}}{=} \textbf{if}\ z = x\ \textbf{then}\ y\ \textbf{else}\ f(z)$
  - 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 abstract the terms to 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., $x \leq y \wedge y \leq x \wedge x \neq y$

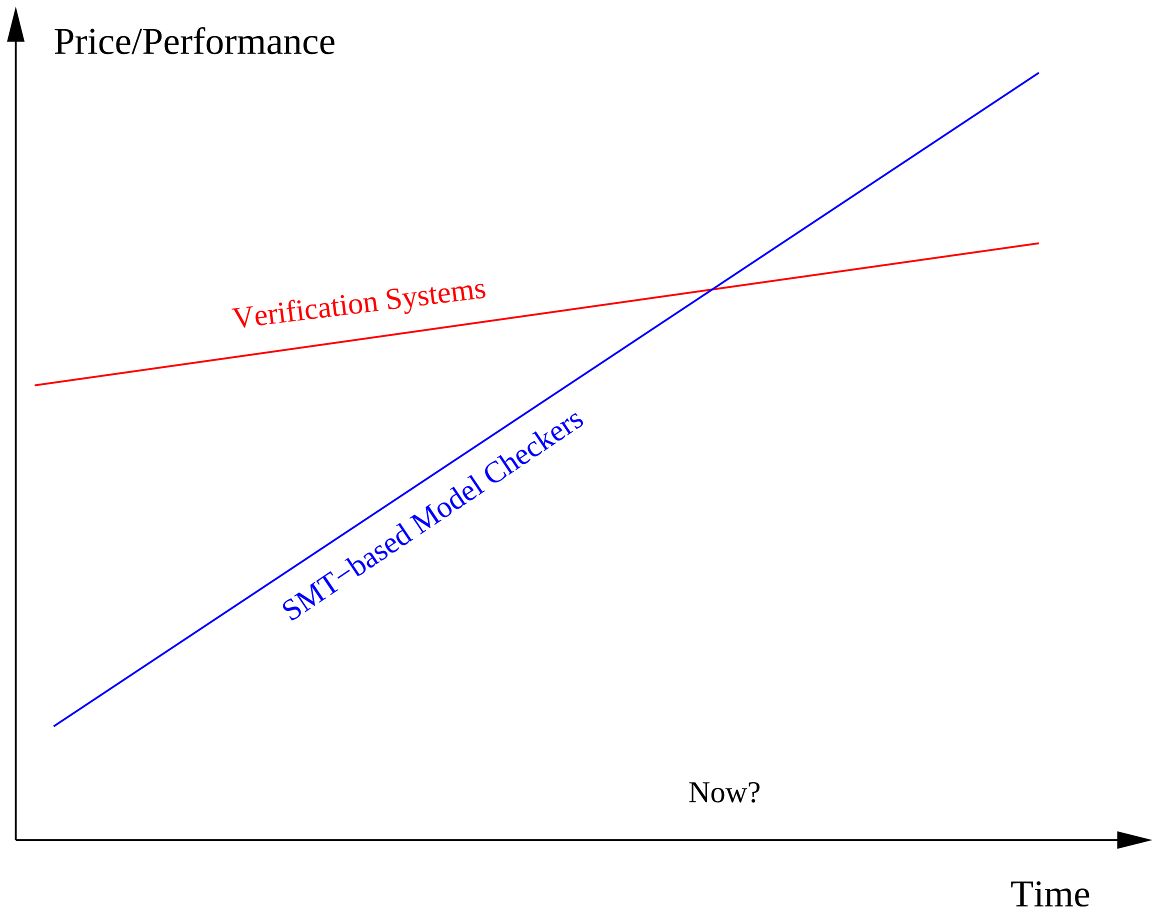
## SMT Solving by "Lemmas On Demand"

- 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

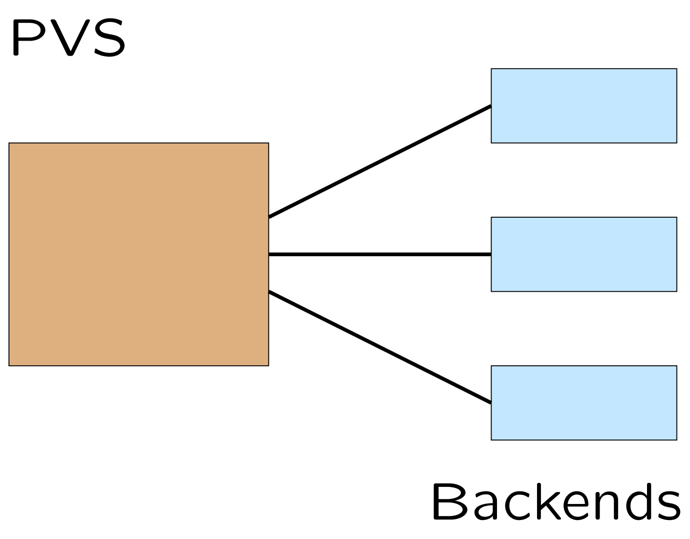
## Fast SMT Solvers

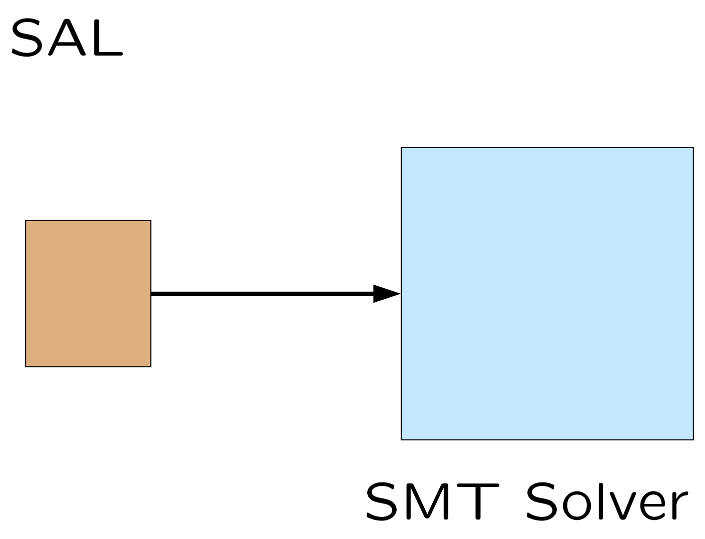
- There are several effective SMT solvers
  - Ours are ICS (released 2002),
    Yices, Simplics (prototypes for next ICS)
  - European examples: Barcelogic, MathSAT
- SMT solvers are being honed by competition
  - Provoked by our benchmarking in 2004
  - Now institutionalized as part of CAV, FLoC

## SMT Solvers as Disruptive Technology



## Verification Systems vs. SMT-Based Model Checkers





Actually, both kinds will coexist via the evidential tool bus

## 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 transition relations)
  - The SAL infinite-state bounded model checker uses an SMT solver (ICS), so handles specifications over reals and integers, uninterpreted functions
  - 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)
  - Note that counterexamples help debug invariant

## 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
- 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
- Is complete for some problems (e.g., timed automata)
  - Fast, too, e.g., Fischer's mutex with 83 processes

## Application: Verification of Real Time Programs

- Continuous time excludes automation by finite state methods
- Timed automata methods 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 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
- 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 (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
  - 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

## Beyond Verification

- Modern formal methods tools do more than verification
- They also do refutation (bug finding)
- And test-case generation
- And controller synthesis
- And construction of abstractions and abstract interpretation
- And generation of invariants
- And . . .
- Observe that these tools can return objects other than verification outcomes
    - Counterexamples, test cases, abstractions, invariants

  Hence, heterogeneous integration

## Integration of Heterogeneous Components

Effective tools are specialized often integrate many components

For example, software model checkers generally have:

- C front end with CFG analyzer
- Predicate abstractor
  - Which uses decision procedures
  - And possibly a model checker
- Model checker and counterexample generator
- Counterexample concretizer and refinement generator
  - Which uses Craig interpolation
  - Or unsat cores

And a control loop around the whole lot

## An Example: LAST

- LAST (Xia, DiVito, Muñoz) generates MC/DC tests for avionics code involving nonlinear arithmetic (with floating point numbers, trigonometric functions etc.)
- Applied it to Boeing autopilot simulator
  - Modules with upto 1,000 lines of C
  - 220 decisions
- Generated tests to (almost) full MC/DC coverage in minutes

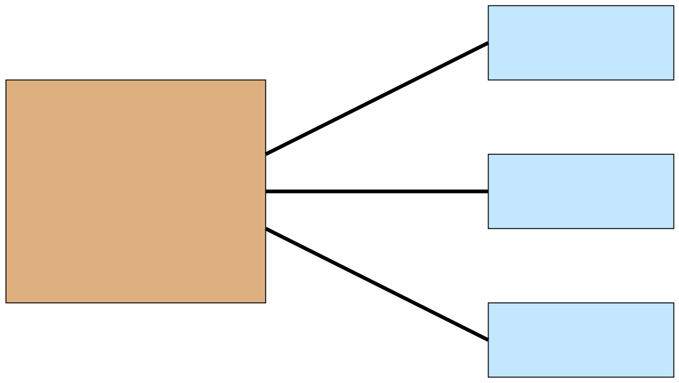
## Structure of LAST

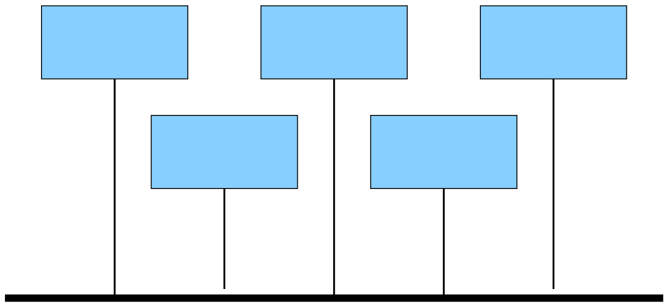
- It's built on Blast (Henzinger et al)
    - A software model checker, itself built of components
    - Including CIL and CVC-Lite
- But extends it to handle nonlinear arithmetic using RealPaver (a numerical nonlinear constraint unsatisfiability checker)
    - Added 1,000 lines to CIL front end for MC/DC
    - Added 2,000 lines to RealPaver to integrate with CVC-Lite (Nelson-Oppen style)
    - Changed 2,000 lines in Blast to tie it all together

## A Tool Bus

- How can we construct these customized combinations and integrations easily and rapidly?
- The integrations are coarse-grained (hundreds, not millions of interactions per analysis), so they do not need to share state
- So we could take the outputs of one tool, massage it suitably and pass it to another and so on
- A combination of XML descriptions, translations, and a scripting language could probably do it
- Suitably engineered, we could call it a tool bus

## From Backends to Bus





- Bus is a federation of equals
- Theorem prover is just another component

## But . . .

- But we'll need an ontology
- The names and capabilities of the tools out there, and how to script the desired interactions
  - Vulnerable to change, if fixed
- Whereas I would like to exploit whatever is out there
  - And in 15 years time there may be lots of things out there
- That is, I want the bus to operate declaratively
  - By implicit invocation
- And I want evidence that supports the overall analysis
  (i.e., the ingredients for a safety or assurance case)
- That is, I want a semantic integration

## A Formal Tool Bus: Logic is the Ontology

- The data manipulated by tools on bus are formulas in logic
- In fact, they can be seen as formulas in a logic
  - The Formal Tool Bus Logic
  - Each tool operates on a sublogic
  - Syntactic differences masked with XML wrappers
- No point in limiting the expressiveness of the tool bus logic
  - Should be at least as expressive as PVS
    - ⋆ Higher order, with predicate, structural, and dependent subtypes, abstract data types, recursive and inductive definitions, parameterized theories, interpretations
  - With structured representations for important cases
    - ⋆ State machines (as in SAL), counterexamples, process algebras, temporal logics . . .
    - ⋆ Handled directly by some tools, can be expanded to underlying semantics for others

## Tool Bus Judgments

The tools on the bus evaluate and construct predicates over expressions in the logic—we call these judgments

**Parser**: A is the AST for string S

**Prettyprinter**: S is the concrete syntax for A

**Typechecker**: A is a well-typed formula

**Finiteness checker**: A is a formula over finite types

**Abstractor to PL**: A is a propositional abstraction for B

**Predicate abstractor**: A is an abstraction for formula B wrt. predicates $\phi$

**GDP**: A is satisfiable

**GDP**: C is a context (state) representing input G

**SMT**: $\rho$ is a satisfying assignment for A

## Tool Bus Queries

- Tools publish their capabilities and the bus uses these to organize answers to queries

  **Query**: `well-typed?(A)`

  **Response**: `PVS-typechecker(...)` $\vdash$ `well-typed?(A)`

  The response includes the exact invocation of the tool concerned

- Queries can include variables

  **Query**: `predicate-abstraction?(a, B,` $\phi$`)`

  **Response**:

  `SAL-abstractor(...)`$\vdash$`predicate-abstraction?(A, B,` $\phi$`)`

  The tool invocation constructs the witness, and returns its handle A

## Tool Bus Operation

- The tool bus operates like a distributed datalog framework, chaining on queries and responses
- Similar to SRI AIC's Open Agent Architecture
    - And maybe similar to MyGrid, Linda, . . . ?
- Can have hints, preferences etc.
- Tools can be local or remote
- Tools can run in parallel, in competition
- The bus needs to integrate with version management

## Scripting

Three levels of scripting

**Tools**:

- Tools should be scriptable
- Better functionality, performance than wrappers
- E.g., SAL model checkers are Scheme scripts over an API
- Test generator is another script over the same API

**Wrappers**:

- Some functionality can be achieved by a little programming and maybe some tool invocation

**Tool Bus**:

- Scripts are chains of judgments

## Tool Bus Scripts

- Example
  - If A is a finite state machine and P a safety property, then a model checker can verify P for A
  - If B is a conservative abstraction of B, then verification of B verifies A
  - If A is a state machine, and B is predicate abstraction for A, then B is conservative for A
- How do we know this is sound?
- And that we can trust the computations performed by the components?

## An Evidential Tool Bus

- Each tool should deliver evidence for its judgments
  - Could be proof objects (independently checkable trail of basic deductions): research topic ’cos raw objects too big
  - Could be reputation (“Proved by PVS”)
  - Could be diversity (“using both ICS and CVC-Lite”)
  - Could be declaration by user
    - ⋆ “Because I say so”
    - ⋆ “By operational experience”
    - ⋆ “By testing”
- And the tool bus assembles these (on demand)
- And the inferences of its own scripts and operations
- To deliver evidence for overall analysis that can be considered in a safety or assurance case—hence evidential tool bus

## The Evidential Tool Bus

- There should be only one evidential tool bus
- Just like only one WWW
- How to do it?
  - Standards committee?
  - Competition and cooperation!
- Probably not difficult to integrate multiple buses
  - Need agreement on ontologies
  - Fairly minimal glue code to link them together
- We'll be building one
  - Initially to integrate PVS and SAL
  - And to reconstruct Hybrid-SAL
- Will appreciate your input, and hope you'll like to use it, and to attach your tools

## Conclusion

- Interesting question is the division of responsibility between the hybrid system core and the supporting computer science
  - E.g., cope with jitter in the control, or eliminate it in the computer science?
- Balanced solutions require knowledge of each technology's requirements and capabilties
- And formal analysis and verification require
  - Automation: SMT solvers as a disruptive technology
  - Integration of each others' tools: an evidential tool bus
- The next challenge may be autonomous systems
  - Top levels use AI planning And diagnosis

  For which assurance may be developed through combinations of online verification, controller synthesis, hybrid systems, . . . and everything else

## Thank You!

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