IARP, IEEE/RAS and EURON meeting, Nagoya, Japan, 16-18 June 2005

# Formal Methods for Dependable Robots In Human-Centered Environments

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

## Overview

- Formal methods in general
  - Refutation as well as verification
  - A small demo
- Formal methods and robotics
- Formal methods and robots in human-centered environments
  - A small demo
- Conclusion

## Formal Methods in General

- Formal methods are about calculating the behavior of computational systems
- Engineers in traditional disciplines build mathematical models of their designs
- And use calculation to establish that the design, in the context of a modeled environment, satisfies its requirements
- Only useful when mechanized (e.g., CFD)
- Used in the design loop: exploration, debugging, testing
  - Model, calculate, interpret, repeat
- Also used in certification: we no longer march armies over bridges to check their strength
  - Instead, verify by calculation that the modeled system satisfies certain requirements

## Formal Methods Are the Applied Math of Computation

- Modeling in engineering uses the applied math appropriate to the discipline concerned
  - Aerodynamics: partial differential equations
  - Computation: mathematical logic
    - ⋆ Doesn't require special notation (although it can help): just reinterpret C, Stateflow/Simulink, etc. as logical specifications rather than instructions to a machine or simulator
- And corresponding efficient methods of calculation
  - Aerodynamics: Navier-Stokes solvers, CFD
  - Computation: automated deduction
    - ⋆ Theorem proving, model checking, constraint solving
    - ⋆ Notice these are symbolic methods

## Benefit of Formal Methods

- A single symbolic calculation subsumes many individual cases
  - Just as $x^2 - y^2 = (x - y) \times (x + y)$
  - Subsumes $36 - 16 = 2 \times 10$ and $49 - 4 = 5 \times 9$ and . . .
  - Price paid is computational complexity
    - ⋆ All automated deduction is exponential or worse
- In comparison with simulation, testing etc., a benefit of formal methods is you can consider all possible behaviors
  - E.g., examined 259,220,300,300,290 states ( $10^{15}$) for TTA startup in a few hours
  - For the modeled system and its environment
- Because formal methods are computationally demanding, may deliberately use models that over- or under-approximate the real thing

## Overapproximate Models

- Modeled system has more behaviors than the real thing
  - E.g., assume a less constrained environment
- Effective for verification of safety properties
- For sound verification, need to be sure that model faithfully represents the design, the design is implemented correctly, the environment is modeled adequately and the formal calculations are performed without error
  - These concerns are identical with traditional methods and are handled similarly

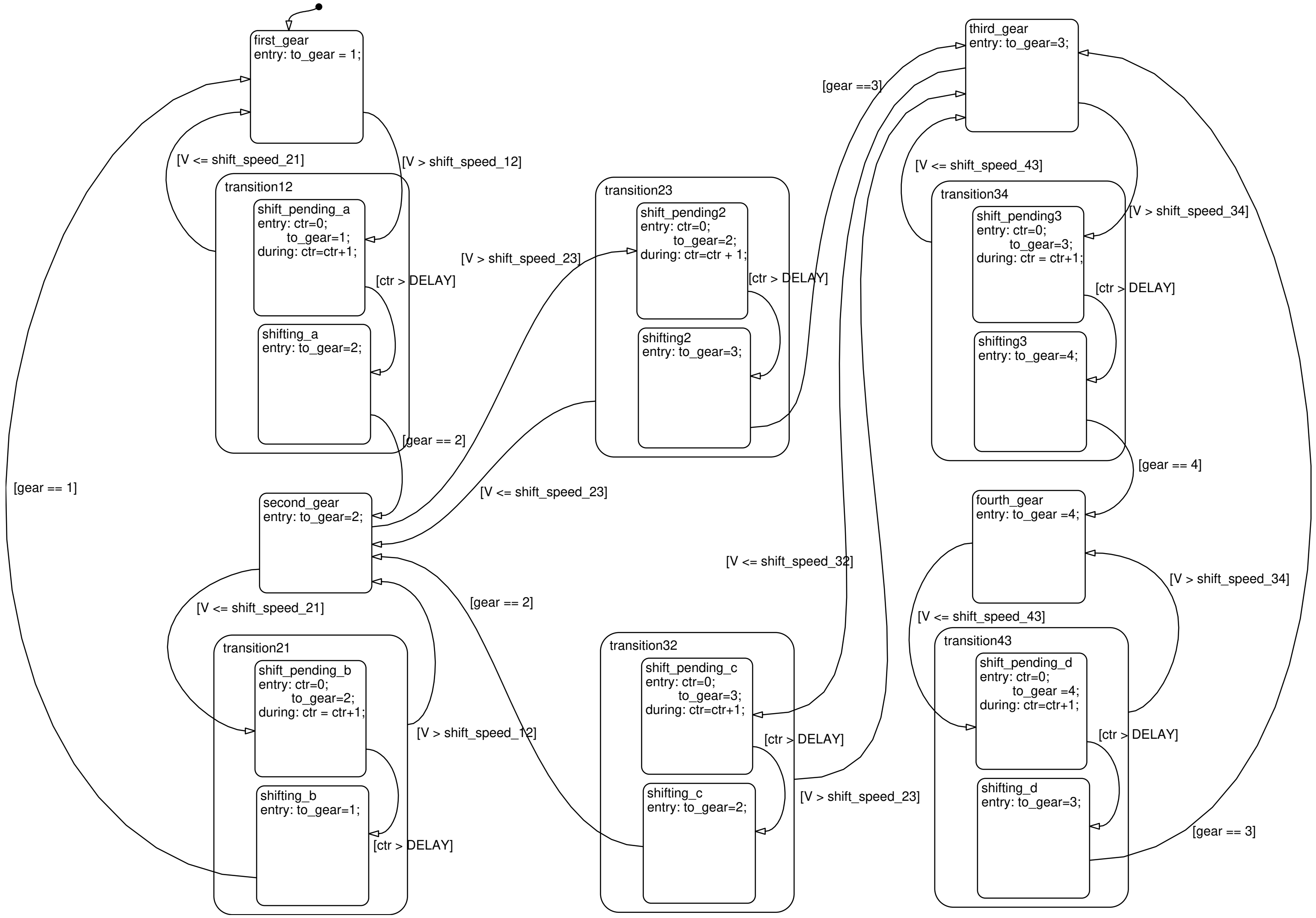
## Underapproximate Models

- Modeled system has fewer behaviors than the real thing
  - Downscale data structures
    - E.g., Limit queues to length 2
  - Limit length of runs considered
  - Constrain the environment
    - E.g., types, numbers of faults allowed
- Risky for verification
- But effective for refutational activities
  - Exploration (experiments to gain understanding)
  - Counterexample generation and debugging
  - Test generation
- Experience is that you find more bugs by complete examination of incomplete models than by partial examination of the full thing

## Let's See An Example

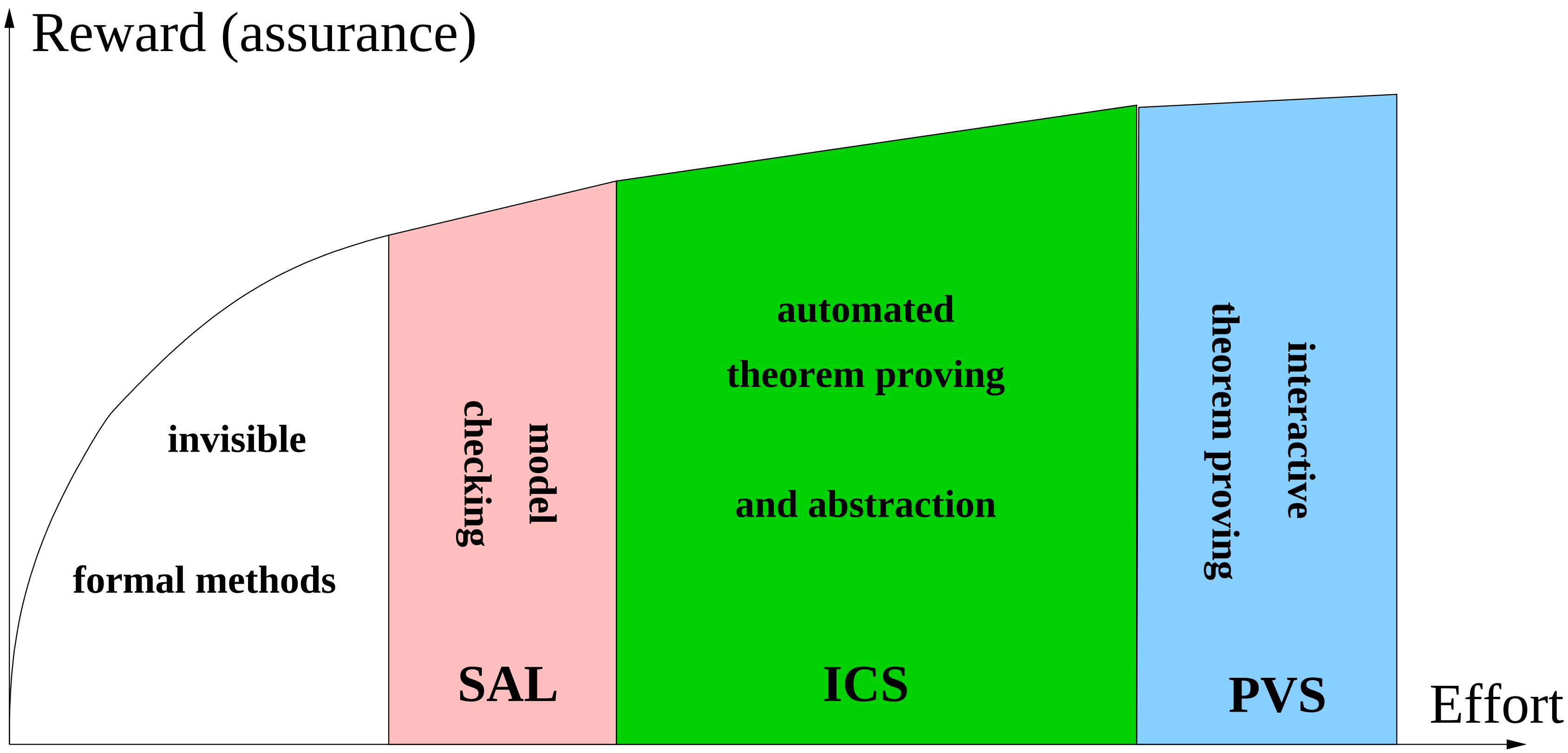
- Given the Matlab/Stateflow description of the shift scheduler for an automatic transmission
  - $3.4832187337996 \times 10^{21}$ reachable states
- Automatically generate efficient test cases to visit every control state and transition, such that the `gear` input changes by exactly one each time (demo just three states for speed)
- Done by automatically translating Stateflow into SAL
  - SAL is the language of our model checkers
- Then using `sal-atg` test generator
  - A Scheme script on the API of our model checkers
  - Uses both symbolic (BDD) and bounded (SAT) model checking

# Stateflow Model for the Shift Scheduler



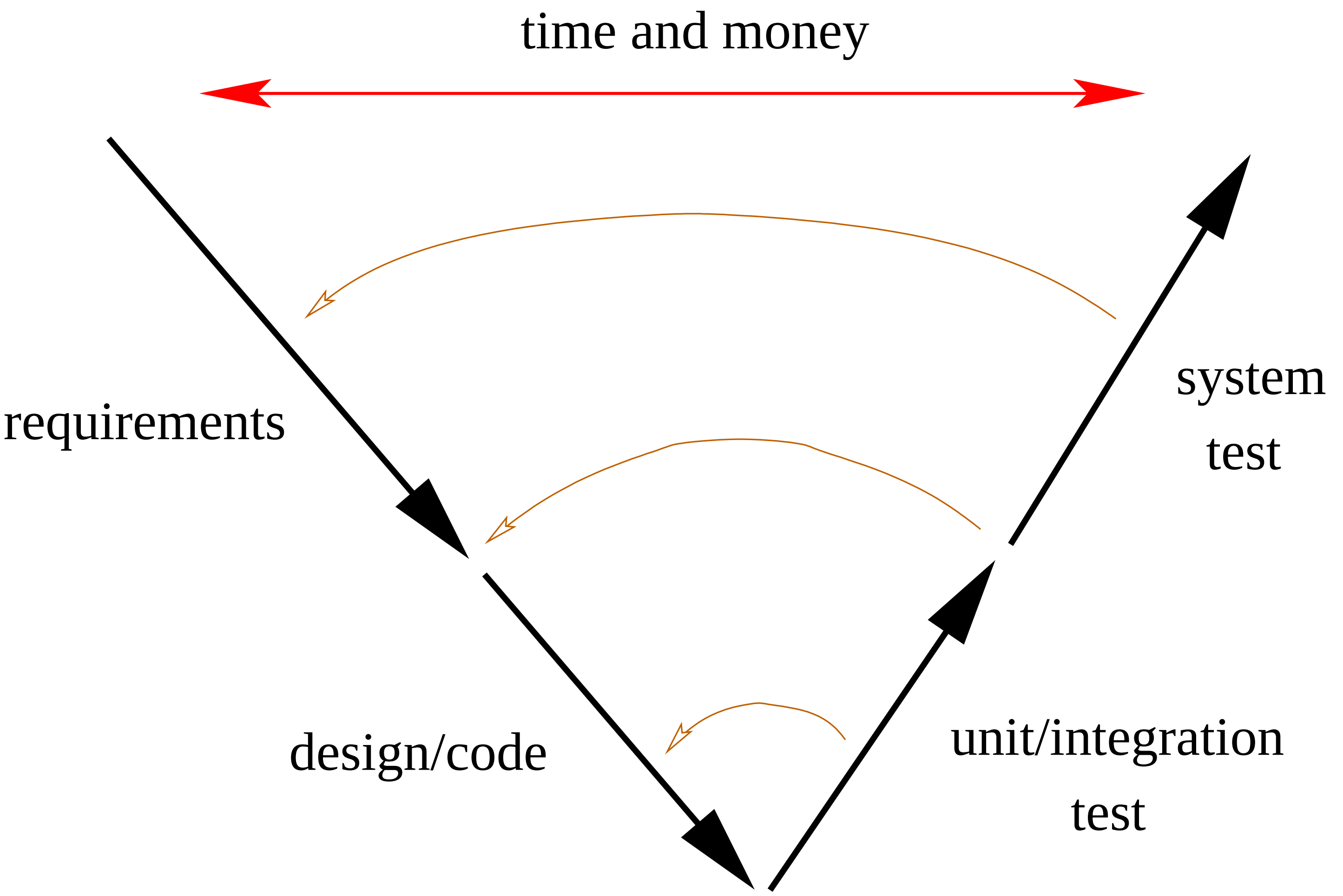
# There is Whole Spectrum of Formal Methods

From invisible (i.e., fully automatic, such as test generation) to full verification by interactive theorem proving



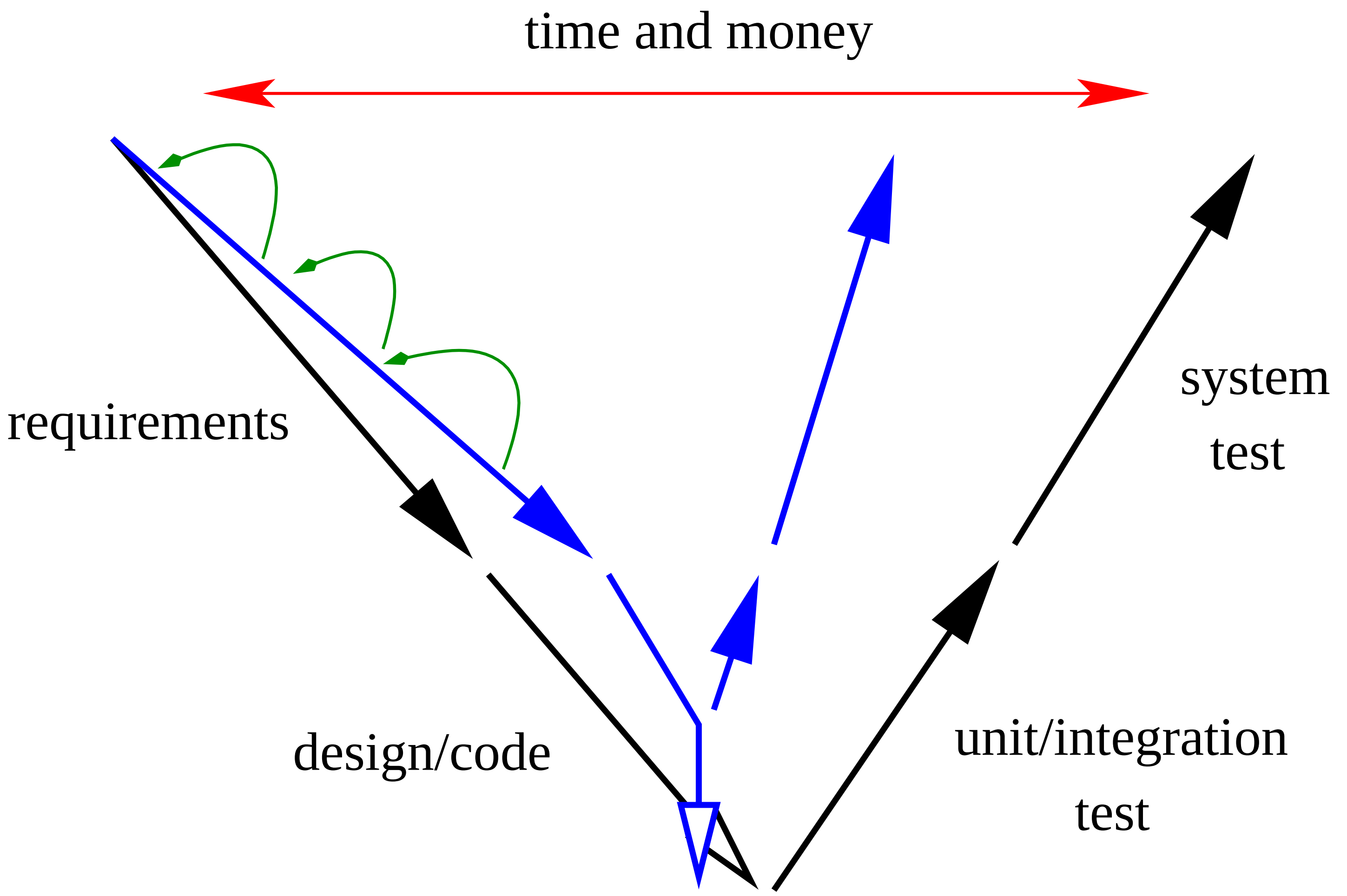

Conjecture: reward/effort climbs steeply in the invisible region

# V&V: Simplified Vee Diagram




Automated formal methods can tighten the vee

# Tightened Vee Diagram




The change to model-based development assists here

## Safety and Formal Methods

- Ability to examine all modeled behaviors is useful when these are numerous or complex
  - Nondeterministic systems
  - Distributed systems
  - Real-time systems
- And can be valuable in safety analysis
- Can be used to automate FMECA and FTA
- And possibly hazard analysis (research topic)
- Also very effective in analyzing fault tolerance

## Progress

- Massive recent improvements in automated deduction
  - Competitions: BDDs, SAT, planners, decision procedures
- And in their use in formal methods
  - Infinite bounded model checking
  - Predicate abstraction
  - Abstraction-refinement
  - Software model checking
  - Runtime verification
- E.g., Microsoft SLAM project

## Formal Methods and Robotics

Several new issues:

- Discrete computation interacting with continuous physics
  - Formally, these are hybrid systems
  - Real-time systems are a special case
- AI elements in the control loop
  - Heuristics, learning, connectionist etc.
- Sensor interpretation
  - Vision etc.

## Hybrid Systems

- State machines plus differential equations
- Control theory can answer questions about stability etc.
- But we are interested in reachability
  - Can we ever reach a bad state?
- There are several formal methods tools for this
  - E.g., HyTech, CheckMate
  - Limited to about 5 continuous variables
- A new method reduces this limitation
  - Hybrid abstraction by Ashish Tiwari (SRI)
  - Can often handle as many as 20 continuous variables
  - Calculates a conservative discrete approximation to the continuous dynamics (a qualitative abstraction)
  - Automated by theorem proving over real closed fields
  - Result is a discrete system that can be model checked

## Hybrid Abstraction: Examples

- Another Ford powertrain example
- 6 continuous variables (and some defined in terms of these)
- 50 polynomials
- About 300 lines of HybridSAL (hand coded from Matlab file)
- Hybrid abstracter takes around 5 minutes (real time) to create the abstractions (about 9,000 lines of SAL)
- SAL symbolic model checker takes about 3 minutes to check abstracted system ($10^{14}$ reachable states)
- Finds a bug: the controller and the plant disagree on the current mode (i.e., gear) of the system
- Proved no chattering: for a fixed throttle position and grade, the transmission does not flip between 1-2-1 or 2-1-2

## AI in the Control Loop

- Scruffy or neat?

  **Scruffy:** ad hoc methods that seem to work

  - E.g., rule-based diagnosis and repair

  **Neat:** methods driven by first principles models

  - E.g., diagnosis from first principles

- To analyze scruffy methods, build suitable models of the controlled plant and environment, and apply formal methods to their combination (models are used for generation)

  - plant + environment + scruffy method $\models$ safety property

  This could be used for runtime verification as well

  - Or Just-In-Time model checking

- Neat methods are correct by construction, relative to the models used (models are used for interpretation)

  - Problem is adequate performance with a decent model

## Controller Synthesis with Formal Methods

- Design of a controller can be seen as a two-player game between the environment and the controller
  - Environment tries to inject disturbances that will take the plant to a bad state
  - Controller tries to prevent the environment having a winning strategy
- Can use formal methods to synthesize controllers
  - Calculate all possible environment moves to find states from which environment can win
  - Calculate a controller strategy that always avoids those states
- Feasible with the power of modern formal methods
  - Use hybrid abstraction for continuous variables
- E.g., SAL plays good endgames in chess, and chess variants

## Sensor Interpretation

- Formal V&V for sophisticated sensor interpretation such as vision seems very difficult: neural systems with learning
- May be able to apply refutational formal methods, and runtime verification
  - world $+$ sensor $+$ interpretation $\models$ safety property
- May be more promising to apply formal methods to fusion of multiple sensors
  - Consistency under conservative models
- And to issues at system level
  - E.g., outermost loop as a controller with noisy sensors
- Example: Berkeley detector for falls by elderly people

## Human-Centered Systems

- Everything is as before, except now the environment includes humans
- So we just need models for relevant aspects of human biomechanics, physiology, cognition etc. and we can use the previous methods!
- Some evidence for plausibility:
  - Hybrid system models in molecular biology
    - ⋆ Symbolic Systems biology
  - Hybrid system models of whole-body systems (e.g., diabetes)
  - CFD for bloodflow in defective arteries
  - Matlab models for biomechanics
  - Formal mental models in cognition

## Modeling Human Cognition

- Modern psychology regards mental processes as information processing
- So maybe we can use formal methods to look at these
- Focus on human error, the dominant cause of aircraft incidents and accidents (70% of accidents)
- Actually, the error is usually bad design, which provokes automation surprise
- Pilots are surprised by the behavior of the automation
  - Or confused about what "mode" it is in
  - "Why did it do that?"
  - "What is it doing now?"
  - "What will it do next?"

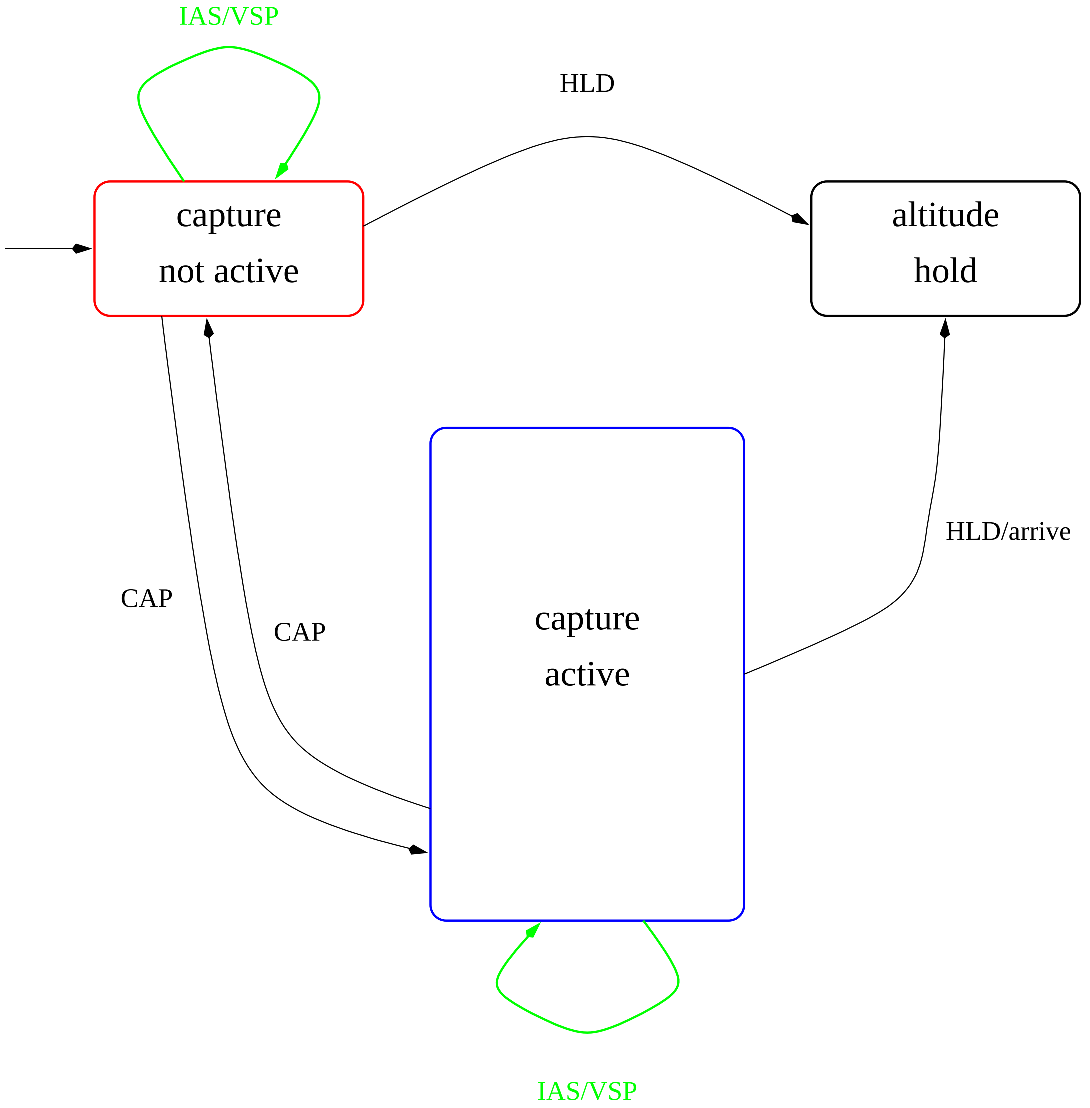
## Postulates (from Human Factors)

- Operators use "mental models" to guide their interaction with automated systems
- Automation surprises arise when the operator's mental model does not accurately reflect the behavior of the actual system
- Mode confusion is a just a special case: the mental model is not an accurate reflection of the actual mode structure
  - Or loses sync with it
- Mental models can be explicitly formulated as state machines
  - And we can "capture" them through observation, interviews, and introspection
  - Or by studying training manuals
    (which are intended to induce specific models)

## Automation Surprise as a Reachability Question

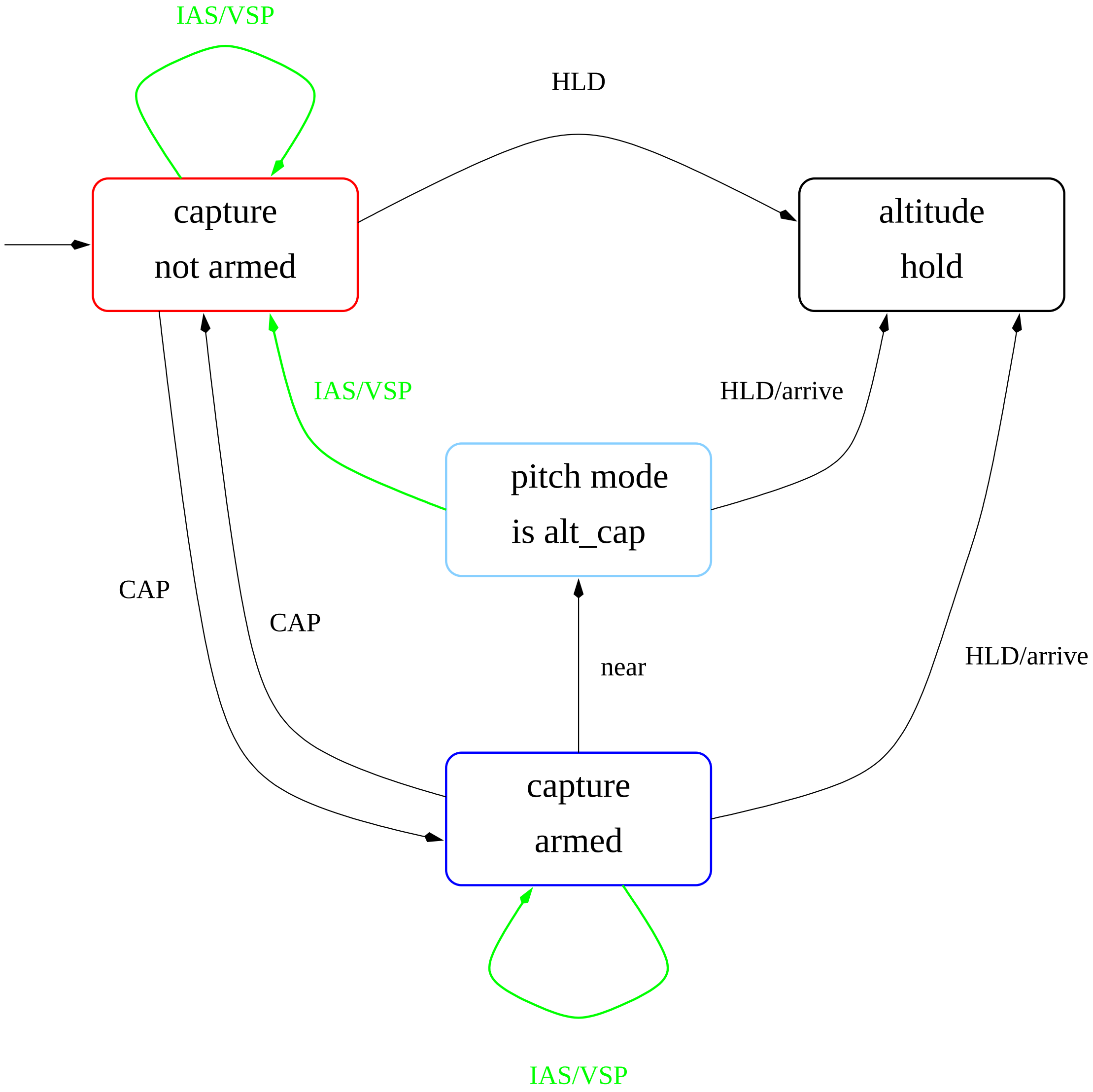
- Take the design of an automated system
  - Represented as a state machine
- And that of a (plausible or actual) mental model
  - Also represented as a state machine
- And ask whether these can ever get out of sync

# Mental Model for Pitch Modes in MD88




Whether capture is active is independent of the pitch mode

# Actual System, Pitch Modes in MD88



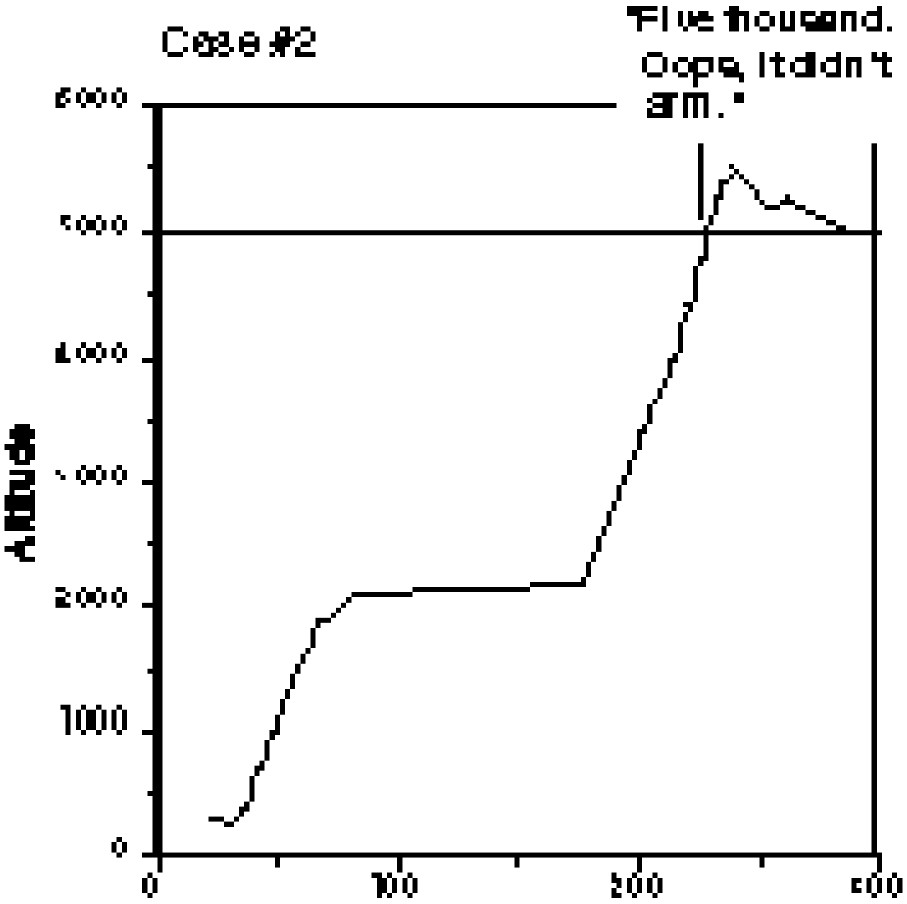

There is an `alt_cap` pitch mode that flies the final capture

## Model Check These

- The actual system is (of course) more complex than the mental model
- But they should agree on the major states
- So model check whether the mental and actual models always agree on whether capture is active
- They do not
- Automatically finds an "automation surprise" scenario that had previously be discovered in flight simulations
- small demo: model checking

## 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



## Other Examples

- Extended this example to examine forgetful pilot and utility of cues supplied by the displays
- We have also used this approach to examine a surprise related to speed protection in A320
- And a known surprise in the pitch modes of the 737 autopilot
- Oishi, Hwang and Tomlin at Stanford has done similar with 777 flap and speed protection management on descent
- Also Ukawa, Ushio and Adachi at Osaka
- Extension: group behavior (multiple pilots, pilots and ATC)
    - Just run multiple models (research topic)

## Conclusion

- Whew!
- "The confidence of ignorance will always overcome the indecision of knowledge"
- Many opportunities for further research
- And interesting collaborations
- SAL, our other formal methods tools, and recent papers (including our Roadmap), are accessible from `http://fm.csl.sri.com`
- My own papers are at `http://www.csl.sri.com/users/rushby/biblio`