
# Hybrid Control Systems Verification

Steve Miller, Darren Cofer
March 4, 2009

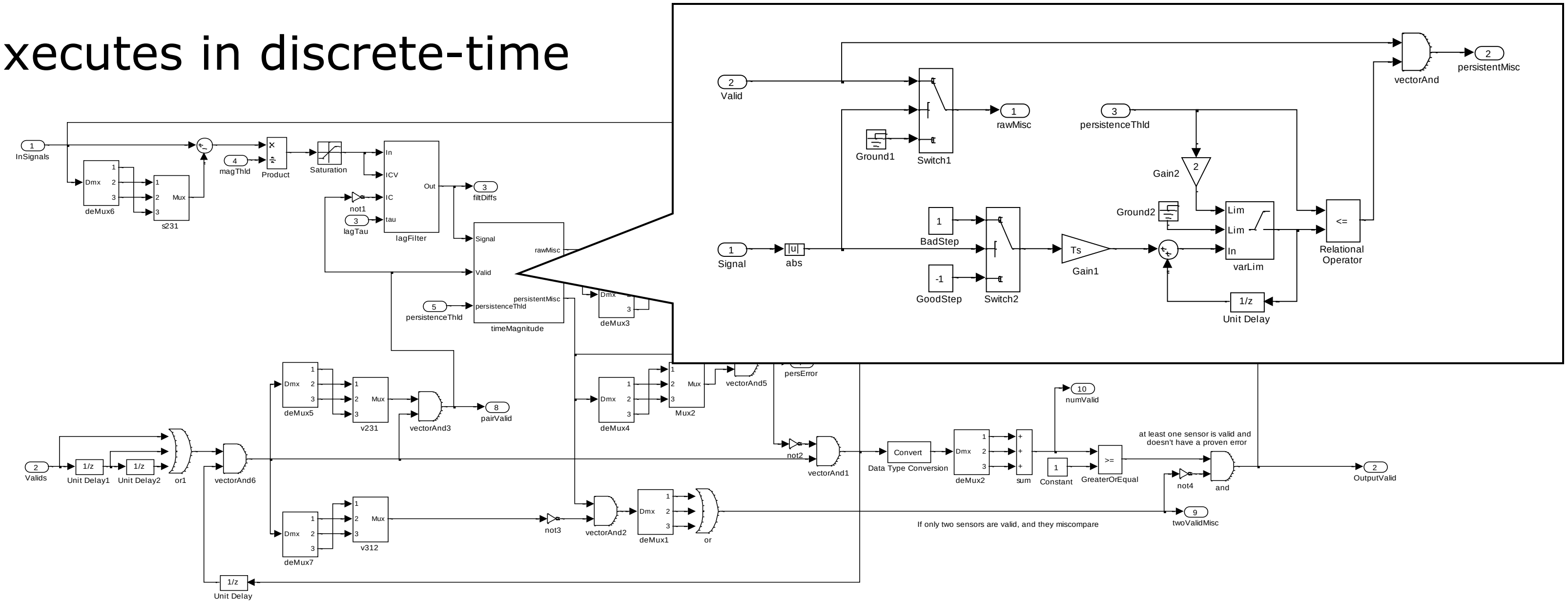
Rockwell Collins

# What kind of hybrid systems are we interested in?

- Product of model-based design
  - built using Simulink/Stateflow or similar MBD environment

***Specific subset of the standard control theoretic definition of HS***

- Combines discrete and continuous dynamics
  - state machines, switches, logic
  - linear/non-linear functions and operators
- Often include floating-point types
- Executes in discrete-time



# Research Challenges We Are Looking for Help With

- Floating point types
- Non-linear arithmetic
- Extracting controller requirements
- Combined methods and tools approaches
- Model checking of asynchronous systems
- Trusted verification tools

# Research Challenges: Floating-Point Types

- **Most Industrial Models Contain Floating-Point Types**
  - Double is the default type for MATLAB Simulink
  - Engineers can use floating-point for simulation and implementation
  - Inability to handle floating-point math is becoming a significant barrier to the acceptance of model checking in industry
- **Theorem Proving of Models with Floating-Point Types**
  - Great work done at Intel, AMD on correctness of FP implementations
  - Focused on bit-level accuracy & small models
  - We are not so concerned with bit-accuracy
    - Prefer to sacrifice completeness rather than soundness
  - Scalability of theorem proving is a major concern for us
    - We have much larger models to verify, e.g.
      Effector Blender: 166 subsystems, 2100+ basic blocks, matrix multiplications, etc.
- **Model Checking of Models with Floating-Point Types**
  - Bit-level integer solvers are impressive and improving:
    works well for fixed-point numbers
  - IEEE 754 provides a clear standard for floating-point math
  - Decision procedures for floating-point types should be possible
  - Probably technically difficult

Ed Clarke:

Decision procedures for real arithmetic that handle floating point arithmetic

# Research Challenges: Non-Linear Arithmetic

- **Non-linear Arithmetic is Common in Industrial Models**
  - Usually appear in conjunction with floating-point types
  - We can translate floating point to real numbers for some problems
    - May hide errors caused by floating point approximations
    - Still useful for debugging rather than proof
- **Some Uses of Real Numbers Are Appropriate**
  - Modeling the real world
  - Aircraft trajectories, plant models, etc
- **Transcendental Functions**
  - Trigonometric functions are common in navigation
  - Need an efficient way to deal with the most common functions sin, arcsin, cos, arcos, tan, arctan, etc.
- **Progress is Being Made**
  - Recent work on non-linear decision procedures for reals [Tiwari]

Ed Clarke:

Decision procedures for real arithmetic that handles floating point arithmetic

# Research Challenges: Extracting Controller Requirements

- **Determining Requirements for Controllers**
  - System requirements are often stated as properties of the combined controller and plant model
  - The plant model is usually larger than controller model, preventing effective analysis
  - Unclear what are the true requirements for the controller model itself

- **Possible Directions**
  - Can useful requirements for the controller be automatically derived from the controller/plant model?
  - Is there an automated way to produce a conservative abstraction of the plant model?
  - Can we use properties proven in the control domain to simplify analysis of the controller?

Rance Cleveland,
Patrick Cousot,
Bruce Krogh:

- Conservatively approximating plant models.
- Plant discretization for discrete analysis.
- Abstracting the plant for open-loop static analysis of the controller.
- Reactivity properties of the closed discrete/continuous loop.
- Assume-guarantee strategies for plant/controller decomposition.
- Extracting requirements from controller models using machine-learning on test data.

# Research Challenges: Combined Approaches

- **Abstract Interpretation tools [Cousot – ASTREE] are very impressive for well-formedness properties**
  - Can Abstract Interpretation be integrated with model checking?
  - E.g., as decision procedures for SMT?
- **How do we leverage multiple tools?**
  - Can theorem proving and model-checking be combined to make the formal verification of complex models tractable?
- **Can we use integer/fixed-point model checking results to make claims about a floating-point implementation?**
  - Replace floating point types in model F with integer/fixed point types to produce model I
  - Under what conditions does a proof about model I tell us something useful about model F?

Ed Clarke, Patrick Cousot, Scott Smolka:

- Combining model checking and abstract interpretation
- Temporal properties and specifications
- Abstraction-refinement alternatives to counter-example guided abstraction refinement

# Research Challenges: Model Checking of Asynchronous Systems

- **Many Industrial Systems are Asynchronous**
  - Components execute synchronously on their own clock
  - System consists of components that communicate via buses
  - Clocks of the components are not synchronous
  - Many systems are quasi-synchronous
    - Nodes have same periods (or multiple of same period)
    - Offset, drift, or jitter may vary between nodes
    - No node can step more than two times before any other node steps once
- **What is the Right Computational Model**
  - Attach clocks to each node?
  - Model synchrony, asynchrony, and quasi-synchrony by constraining the clocks?
- **What are the Right Verification Engines**
  - SMV and NuSMV work well for synchronous models
  - SPIN works well for some asynchronous models
  - Can we choose or tune a model checker based on the type and degree of asynchrony?

Patrick Cousot:

- Extension of Astree for parallel programs running on ARINC 653
- Abstract interpretation for verification of quasi-synchronous parallel execution of synchronous programs

# Research Challenges: Trusted Verification Tools

- **Need Supported Tools**
  - Open source is fine
  - Tool is distinct (separate install) from other tools
  - Doesn't become a part of our products
  - Our evaluation period may last for years
  - Need to be able to license for commercial use
- **DO-178B/C Qualification**
  - Major selling point for civil avionics developers
  - Not that hard for verification tools
    - Need tool operational requirements
    - Need set of tests that show tool satisfies its requirements
  - We know how to do this!
- **More Formal Paths to Trusted Tools**
  - Proof checker
  - Redundant tools
  - Formally verified tools

New topic not discussed in our earlier teleconference.

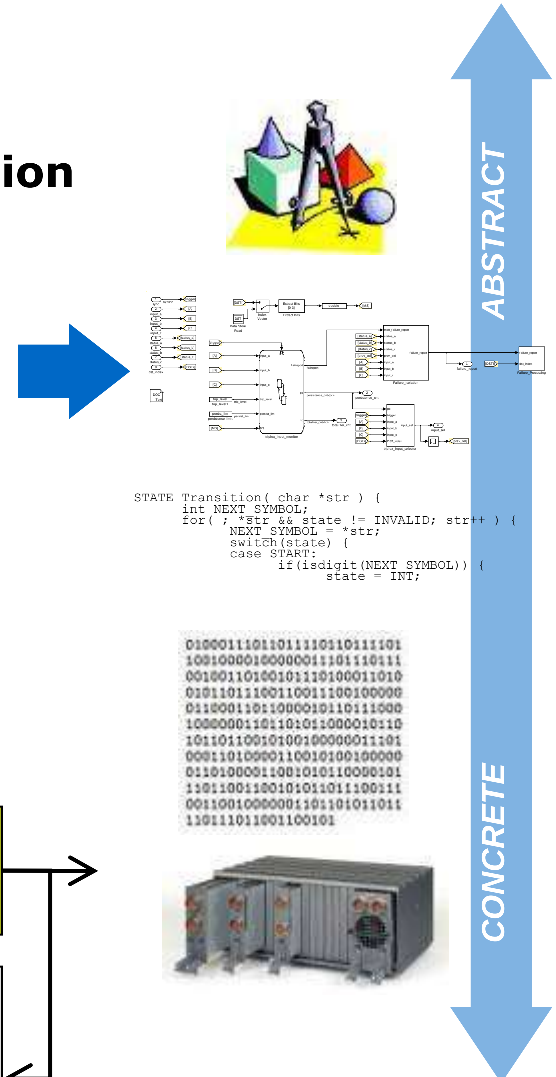
# Background Slides

# Outline

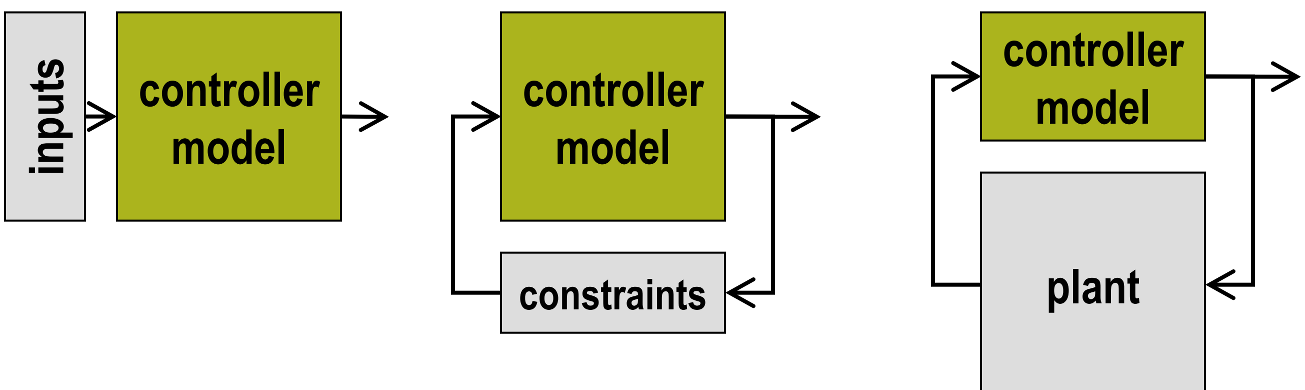
- What kind of hybrid systems are we interested in?
- What are we trying to do?
- What can we do now?
- What are the limitations?
- What have we tried?

# What are we trying to do?

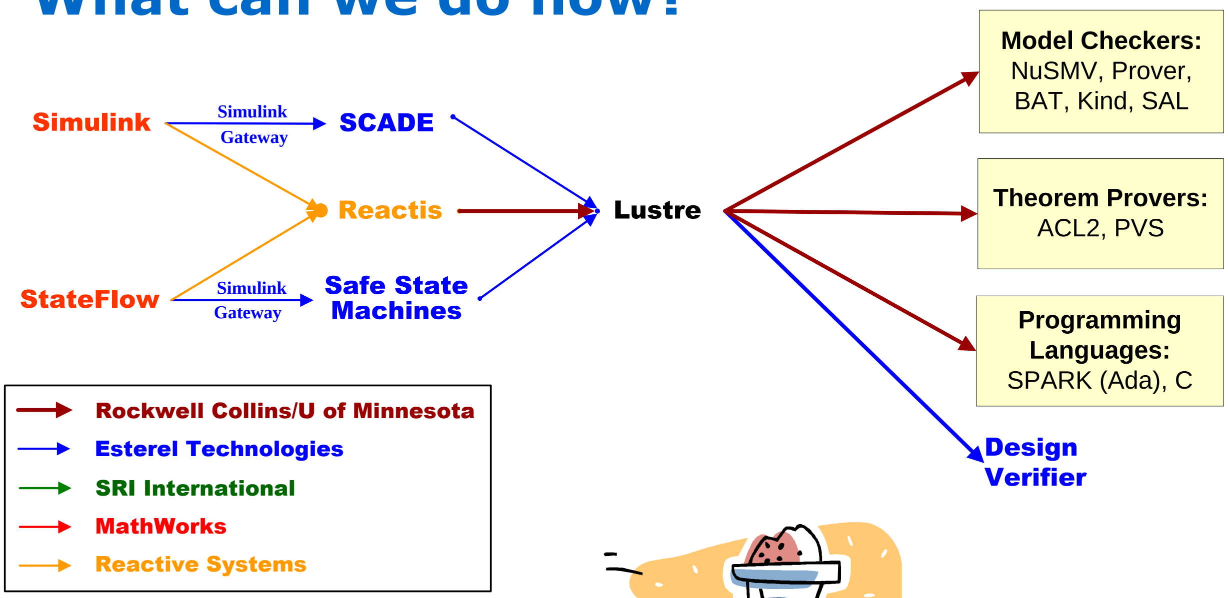
- **Detailed model of controller**
  - **Should correspond closely to implementation**
  - **Verify model correctness with respect to requirements**
- **Simple model of plant/environment**
  - **Provide constraints on input values**
  - **Limit complexity of analysis**
- **Verification of software requirements**
  - **vs. closed-loop behavior of system**




*This…* *…or this* *Not this*

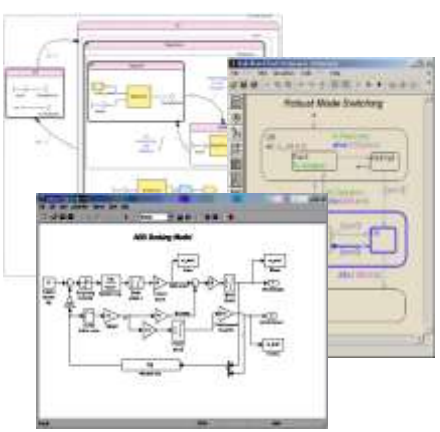


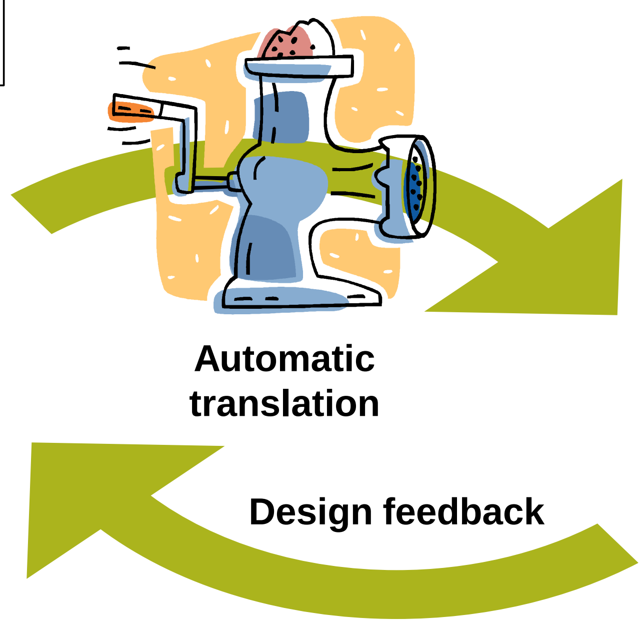
# What can we do now?




## *Gryphon translation framework*

- Supports a wide variety of back end tools and languages
- Straightforward to add new tools (e.g. Prover support: 4 days effort
- Apply "the right tool for the job"





# What are the limitations?

- **This approach works well for certain classes of systems…**
  - Very well for discrete types (Booleans, enumerated types)
  - Can handle a restricted number of integers
  - Real numbers with linear operations
- **…and scales to very large models**
  - 16,117 Simulink blocks, 4,295 subsystems, 1.5 $10^{37}$ reachable states
- **But we don't have a general strategy to handle "numerically intensive" hybrid systems**
  - Analyzing non-linear operations
    - Most industrial models have non-linear operations
    - Most model checkers do not natively support non-linear arithmetic over reals
    - Theorem provers require skilled users and are labor intensive
  - Floating-Point math
    - Floats ≠ Reals (run-time errors)
    - Floating point arithmetic is notoriously difficult to analyze
  - Scaling the analysis
    - Much more difficult to scale model checkers on numeric models

# What have we tried?

- **Predicate abstraction**
  - Suitable for models with a handful of non-linear constraints
  - Add model invariants to exclude spurious counterexamples
  - Offline “counterexample-driven refinement”
- **Model linearization**
  - Arbitrary non-linear calculations are difficult
  - Need to use piecewise approximation
  - Requires significant amount of analysis
  - Approximation leads to incompleteness & unsoundness
- **Fixed-point representation of data**
  - Fixed point numbers can be represented as integers
    - same cost as linear integer operations
      (e.g., 2 divide-by-constants, 1 MOD-by-constant, 2 multiplications-by-variable)
  - Model checkers allow non-linear operations on bit-level integers
  - Pros
    - Can analyze models “as is” without changes
    - Gets around non-linear limitations of decision procedures
    - Can achieve soundness using intervals
  - Cons
    - Need bounds for all vars, scalability limits, incomplete (intervals diverge)

# Summary

- **Our wish list:**
  - We want to analyze large, non-linear numeric models
  - Bit-level accuracy is often not a concern
    - We would rather sacrifice completeness than soundness
    - But, for now, any rigorous analytic tool would be helpful
  - Approximations (such as interval arithmetic) will probably be necessary
  - Can we have “adjustable” accuracy?
    - Higher accuracy likely means lower performance
  - Work on notations and methodologies for control system requirements

- **What we can provide:**
  - Some example problems
  - Feedback on proposed tools and algorithms
  - Ability to translate Simulink/Stateflow models into a variety of back-end notations.
  - Experience applying model checkers and theorem provers on industrial-sized problems