Formal Composition for Time-Triggered Systems

John Rushby and Ashish Tiwari

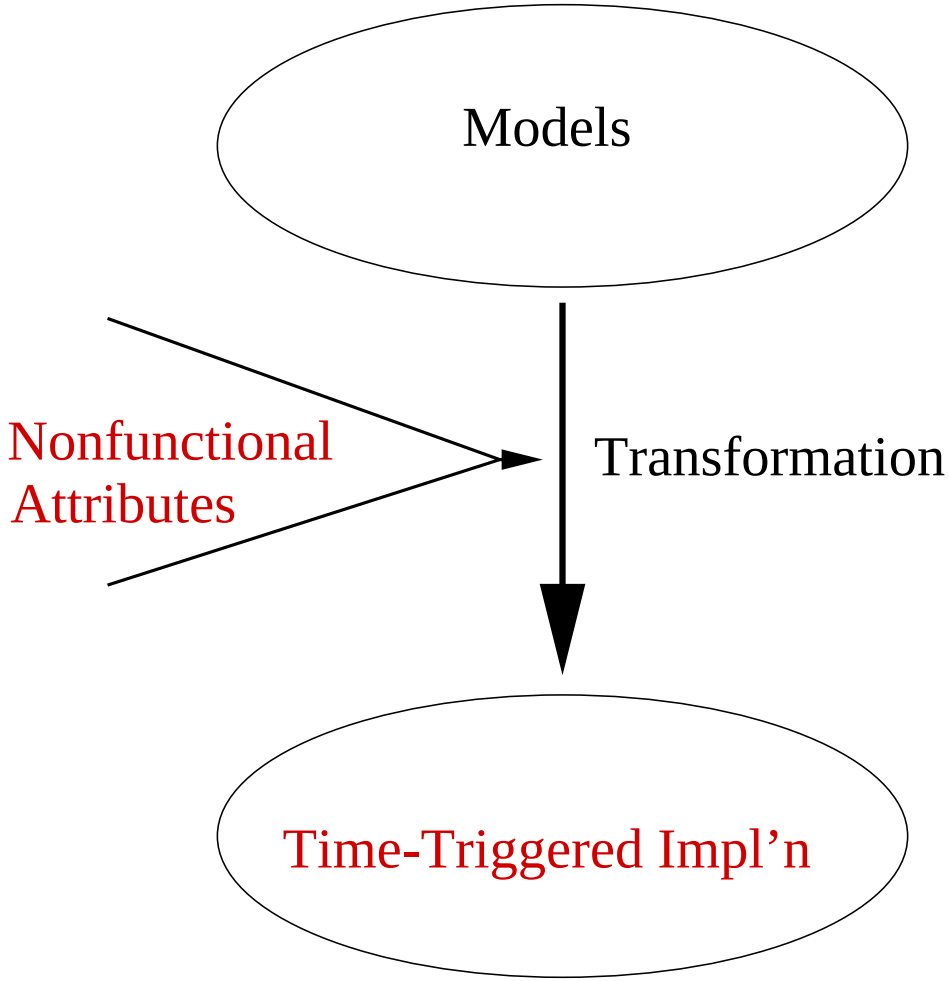
Rushby,Tiwari@csl.sri.com

Computer Science Laboratory

SRI International

Menlo Park CA 94025

Objective: Generic Form



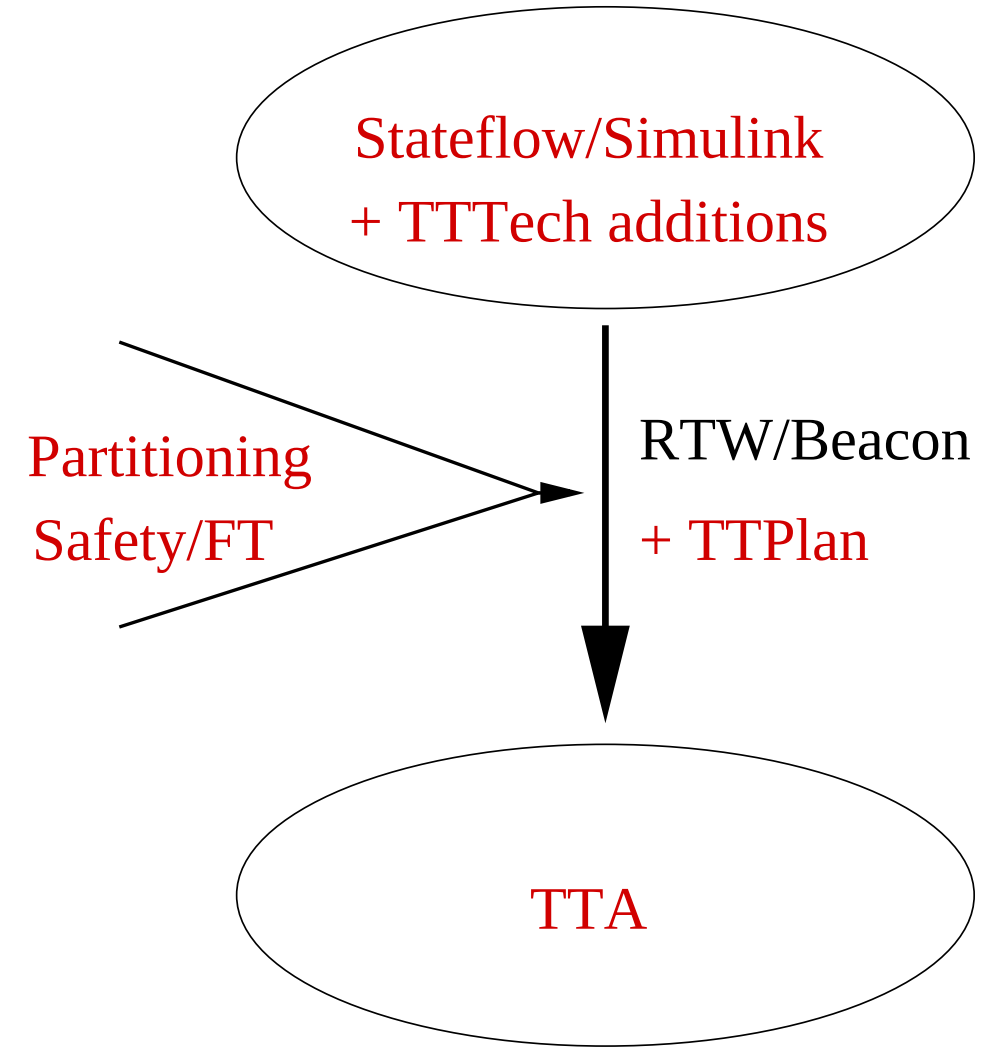
We do

Assurance

Based on mechanized

Formal Methods

Objective: Specific Form



Analyze models

Verify transformations

Verify TT services

To yield assurance for final system

Approach

Background:

- We are the developers of PVS, a system for formal verification (used in about 300 sites)
- We are making its decision procedures available as ICS
- And are building a system SAL that integrates PVS and several other tools such as model checkers

Here, in year 1, we focus on

- Formal semantics and assurance for Stateflow/Simulink
- And design and assurance issues in Time-Triggered Systems

Progress: Semantics and Assurance

For Stateflow/Simulink

- Formal semantics for Stateflow based on translation to Extended Communicating Pushdown Automata (CPDA)

  And mechanization in SAL

  - Can check for bounded stack depth
  - Can model check
- Method for symbolic analysis of Simulink based on cylindrical algebraic decomposition (CAD)

  And mechanization in SAL (using qepcad)

  - Can push symbolic values through a Simulink model
- And the integration of these

AFTI-F16 Departure on Flight Test 36

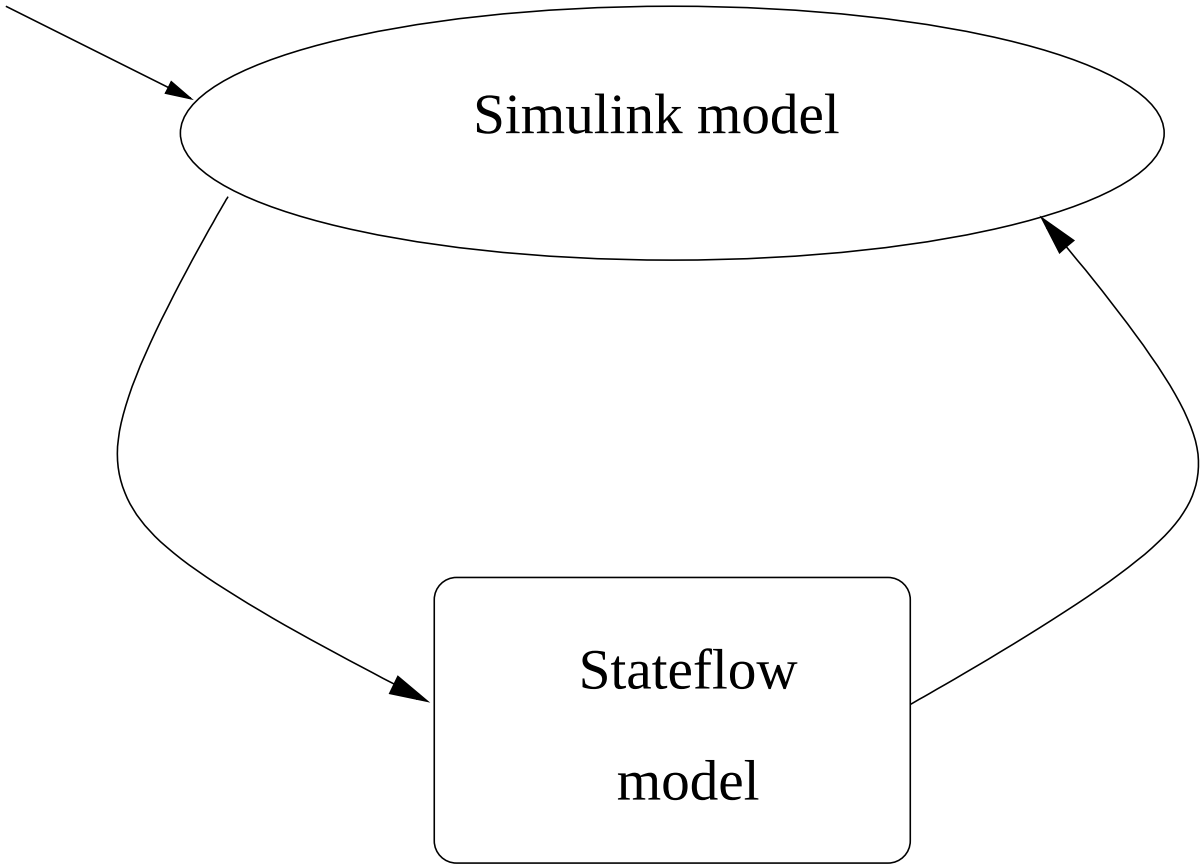
Sideslip exceeded 20°, normal acceleration exceeded first -4g, then $+7$g, angle of attack went to -10°, then $+20^\circ$, the aircraft rolled 360°, the vertical tail exceeded design load, all control surfaces were operating at rate limits, and failure indications were received from the hydraulics and canard actuators.

The side air-data probe was blanked by the canard at the high angle of attack and sideslip achieved during the excursion; the wide input threshold passed the incorrect value through, and different channels took different paths through the control laws

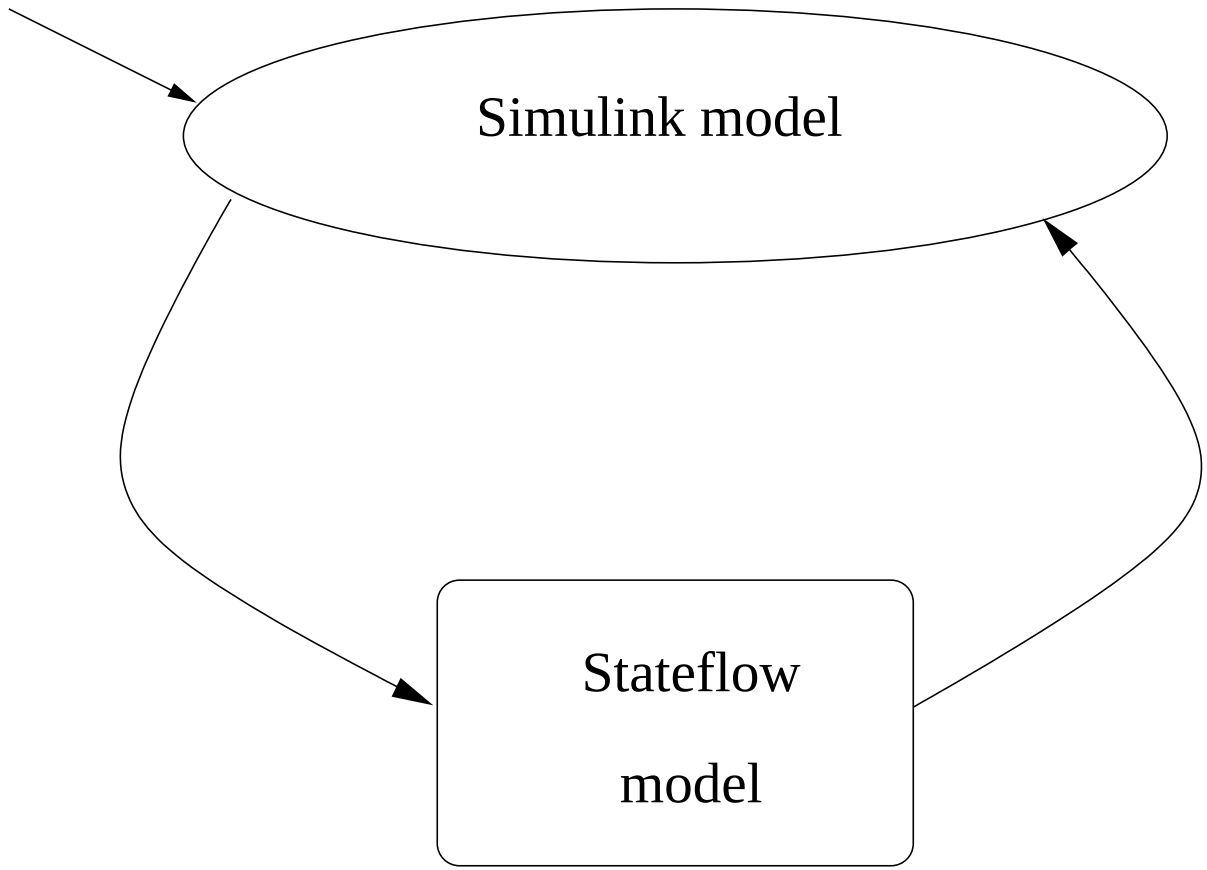
Analysis showed this would have caused complete failure of the DFCS for several areas of the flight envelope

Hence, interested in discrete mode changes in continuous systems

Analyzing Stateflow/Simulink Models



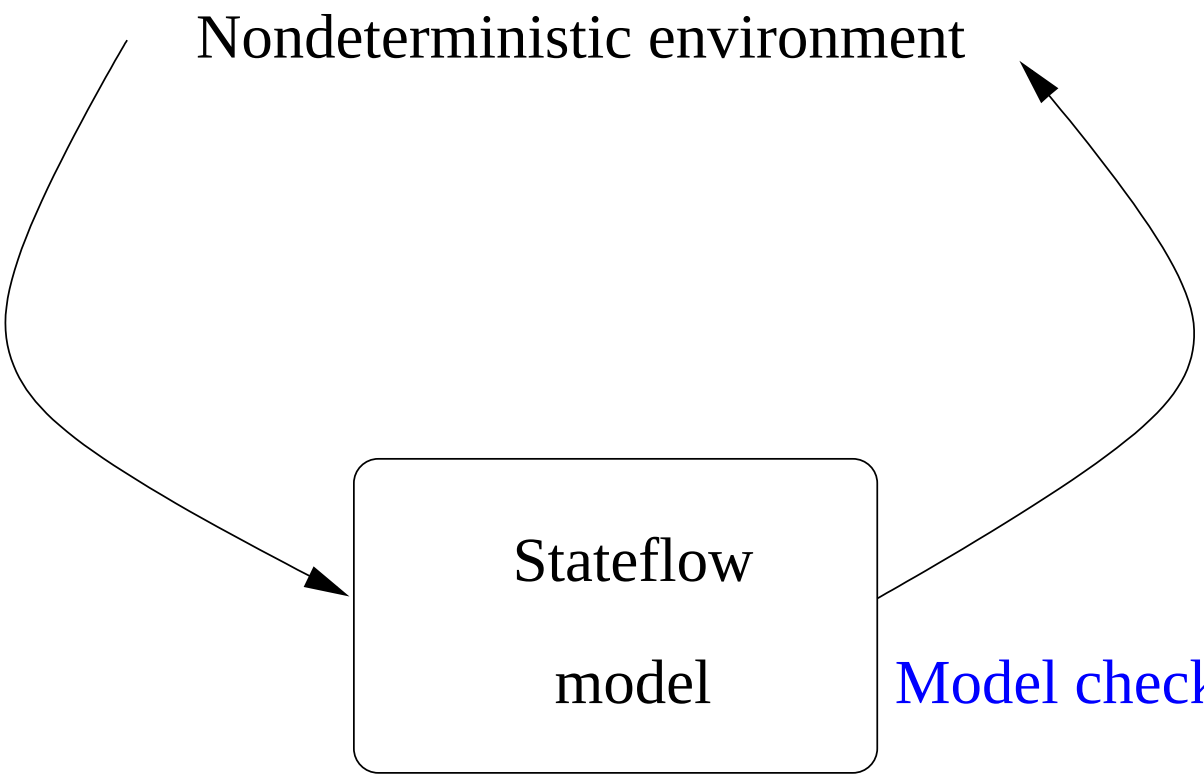
Simulate One Trajectory at a Time



Just like testing: when have you done enough?

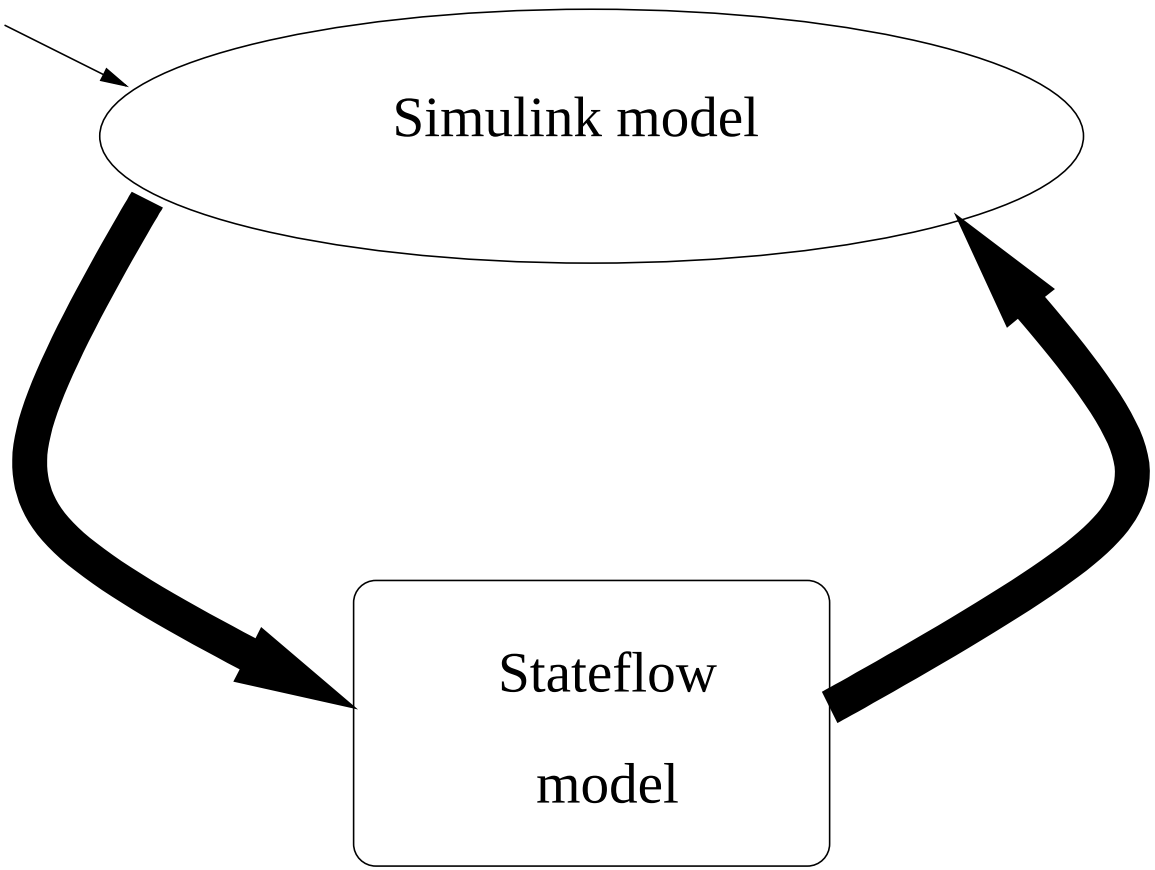
Model Check With

Nondeterministic Environment



Too crude to establish useful properties

Simulate “Envelope” of Trajectories

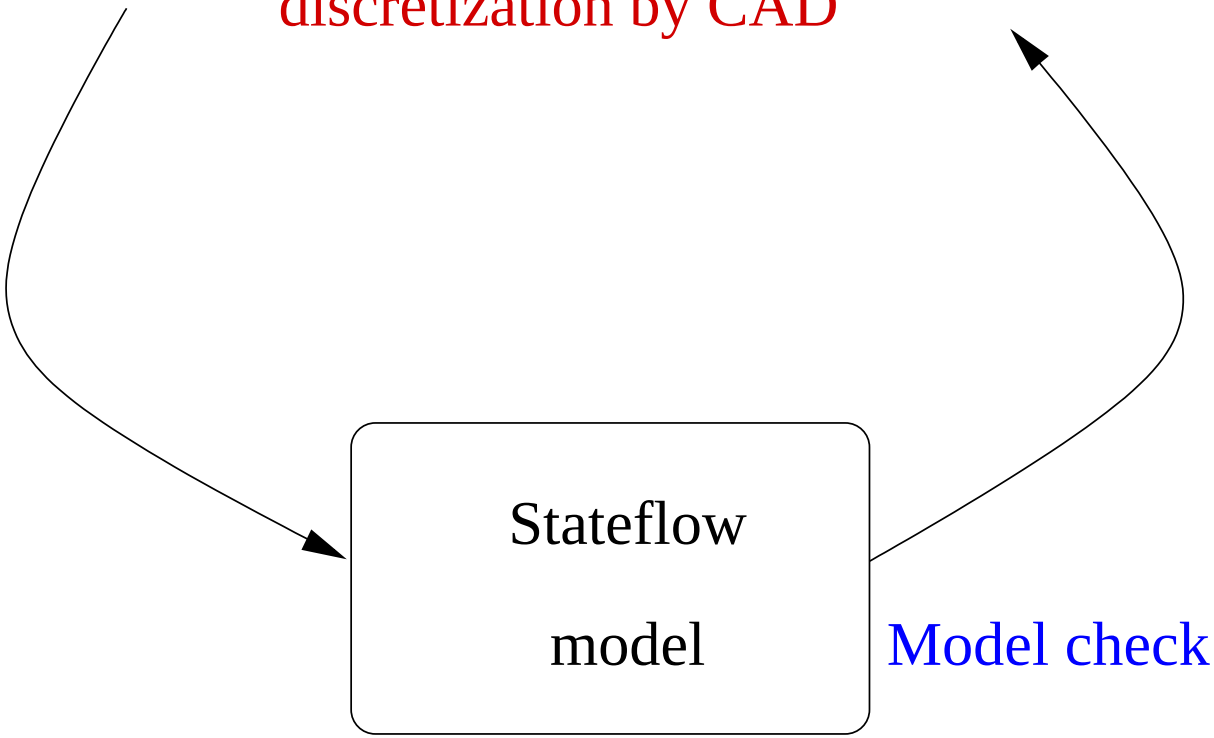


Data or flow pipes do this

Model Check With Environment Props

Established by Symbolic Evaluation

Properties established of a discretization by CAD



Progress: Semantics and Assurance

For Stateflow/Simulink

Assuming the discretizations are conservative:

- Can prove the simple version of the state consistency challenge
- Can prove simplified collision avoidance controller maintains positive gap

Useful for debugging even if not conservative

- Can determine bounded stack depth

Progress: Design and Assurance Issues

In Time-Triggered Implementations

- Have formally verified most of the key algorithms of TTA (NASA project): clock synchronization, group membership, guardian window timing
  - Remaining: clique avoidance, integration
- Developed detailed technical comparison of four commercial time-triggered avionics and automotive buses
  (Safebus, Spider, TTA, FlexRay) available at
  `http://www.csl.sri.com/~rushby/papers/buscompare.pdf` (draft)
- Contemplating an introduction to time-triggered systems

Tech Transfer and OEPs

- Applying all this to Honeywell project developing new-generation FADEC built on TTA for a NATO aircraft
- Formal Stateflow and Simulink analysis available (soon) to any OEP tool suite using XML (if it runs on Linux)
  - Extending to other Statechart variants
  - E.g., UML (in collaboration with TU Budapest)

  Can also tackle challenge problems at the modeling level
- Time-triggered design needs more proselytizing

Part II

- Progress
- Plan

Progress

- Semantics for Stateflow in SAL
- Stateflow model analysis
- Simulink to SAL
- Analysis for Simulink models

Semantics for Stateflow in SAL

The translation consists of two basic steps:

- Translating a Stateflow chart into extended communicating pushdown automata (CPDA).
- Translating the extended CPDA into SAL.

Semantics of a Stateflow transition is obtained in terms of a SAL transition

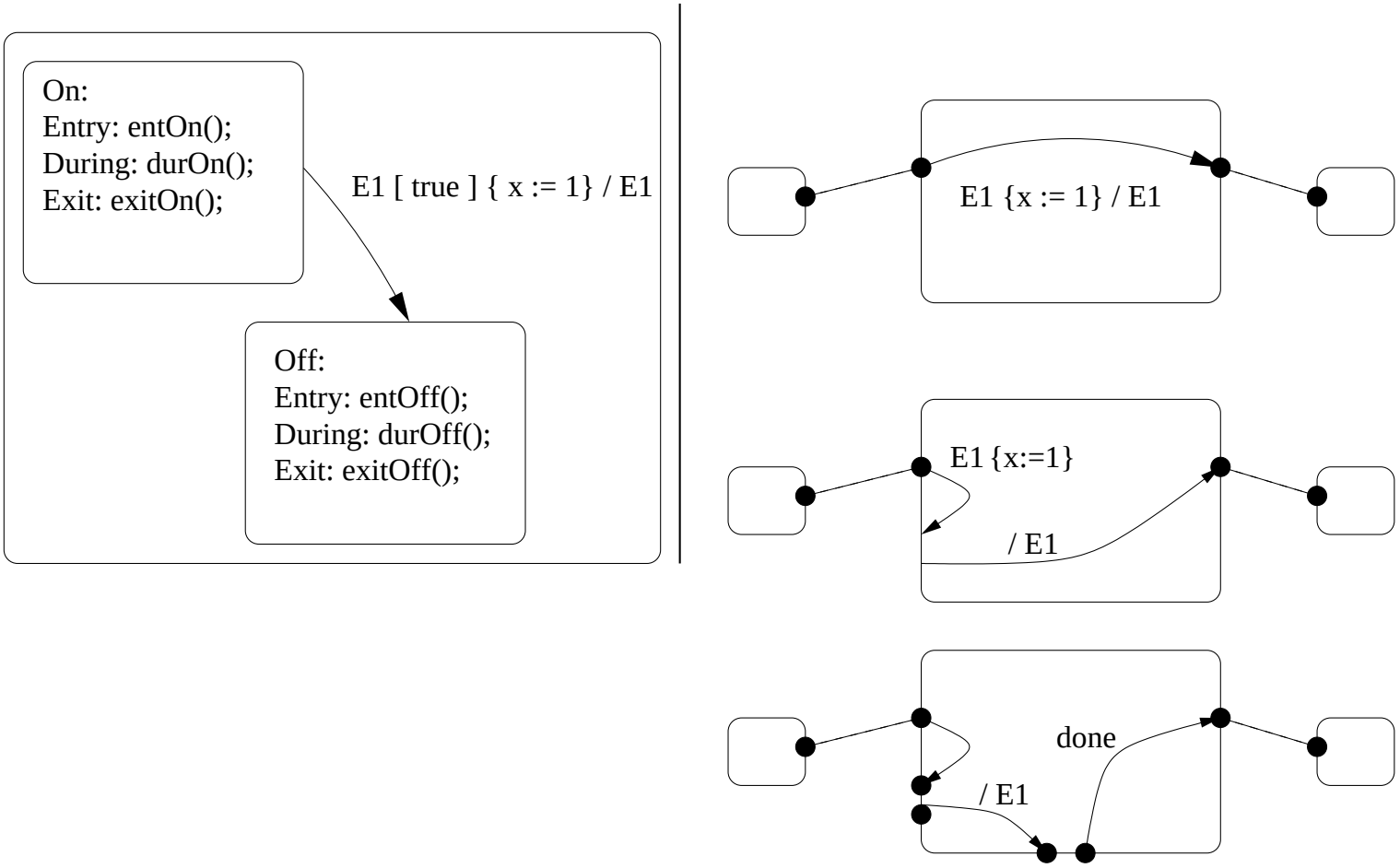
Translation handles hierarchy, nested event broadcasting, junctions, and supertransitions.

Stateflow to CPDA

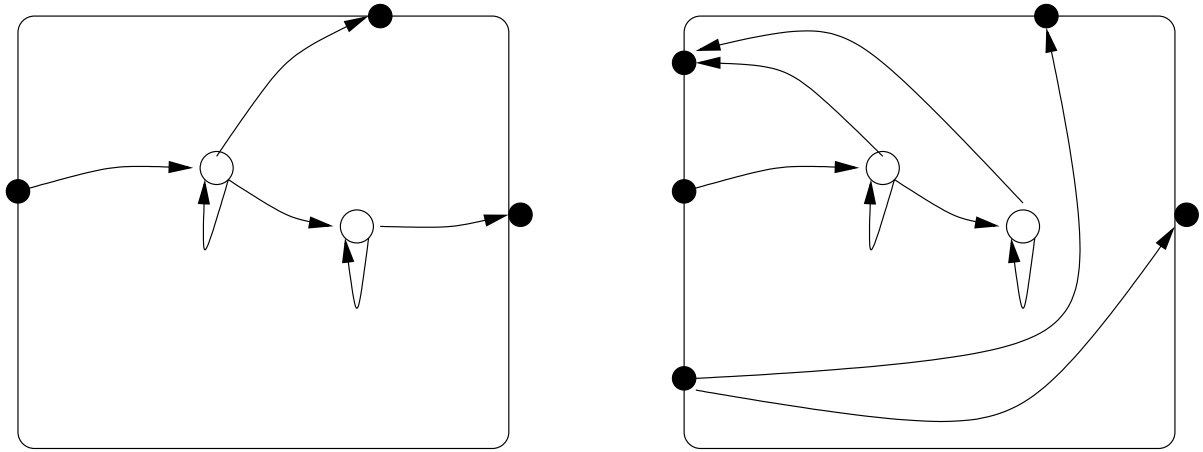
- Automata : State machines in Stateflow
- Stack : nested event broadcasting (infinite loops possible)

  Stack holds history of control flow and partially processed events
- Communication : inter-level transitions, hierarchy

Example: Translation



Example: Flow Diagram Notation Translation



Features of Translation

Features:

- modularity preservation: automata $\leftrightarrow$ SAL module.
- Stateflow transition map to multiple SAL transitions.
- communication between automata $\leftrightarrow$ module composition with variable renaming.

Sample Transitions

```
conFI6 AND preIn12 AND NOT state14 AND true -->
                   preOu13' = TRUE;
                   oport15' = TRUE;
                   conFI6' = FALSE
conFI6 AND preIn12 AND state14 AND true -->
                   ....
conFI6 AND defIn16 AND NOT state14 AND true -->
                   ....
conFI6 AND NOT defIn16 AND NOT preIn12 AND NOT state14 -->
                   ....
conFI6 AND state14 AND NOT defIn16 AND NOT preIn12 -->
                   ....
```

Sample Transitions

```
iport24 AND state2 AND traLo3 AND top(stack) = e0 -->
                   push(e9);
                   oport26' = TRUE;
                   oport7' = TRUE;
conBI8 AND iport27 AND state2 AND top(stack) = e9 -->
                   x' = OR(x, true);
                   stack' = push(e10, S1, e0, stack);
                   oport28' = TRUE;
```

Progress

- Semantics for Stateflow in SAL
- Stateflow model analysis
- Simulink to SAL
- Analysis for Simulink models

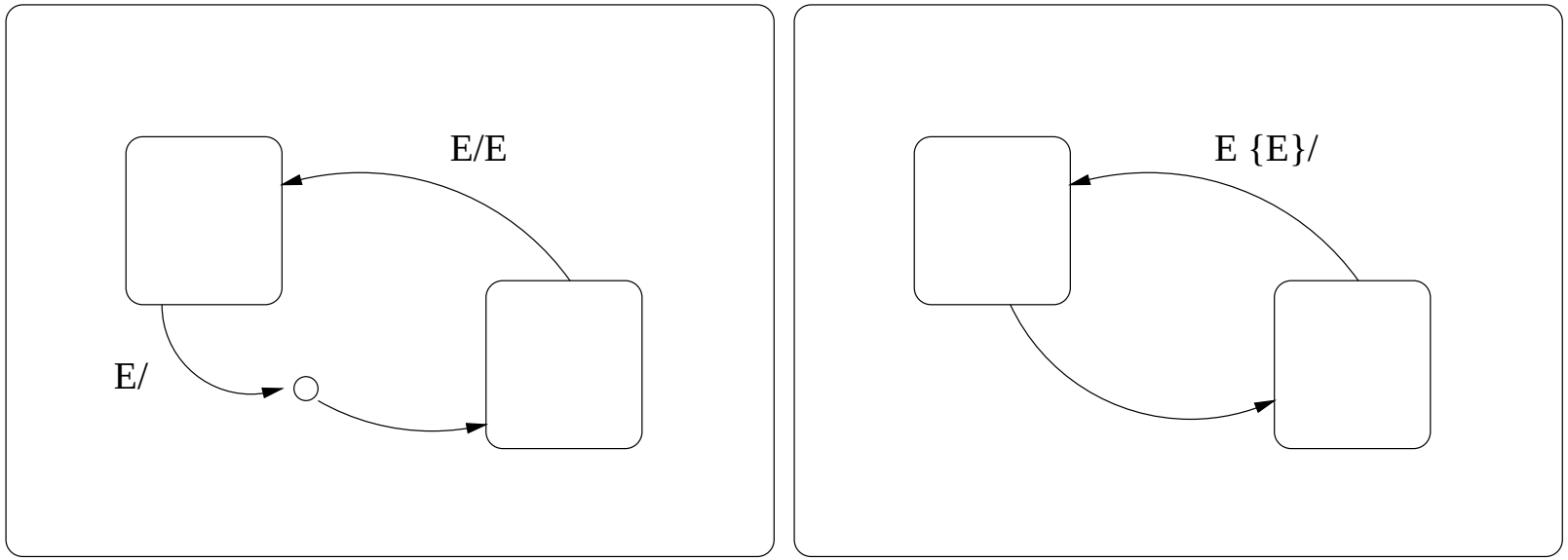
Stateflow Model Analysis

- reachability
- detecting event loops
- model checking

Theorem proving for verifying data invariants and type correctness.

Bounded Stack Depth

The first Stateflow model needs only a bounded stack whereas the second one contains a loop:



Bounded Stack Depth

Approach:

- Construct a Multi-Finite Automata that accepts all states reachable from some starting state
- Check for a loop in the Multi-Finite Automata

Observation: $post^*(\phi)$ is "regular" if ϕ is "regular" in case of transition systems specified by PDAs.

Translation to SAL and bounded depth check is written in Java.

State Consistency Challenge

Observation: Controllers for hybrid systems often replicate and track the modes of the physical system.

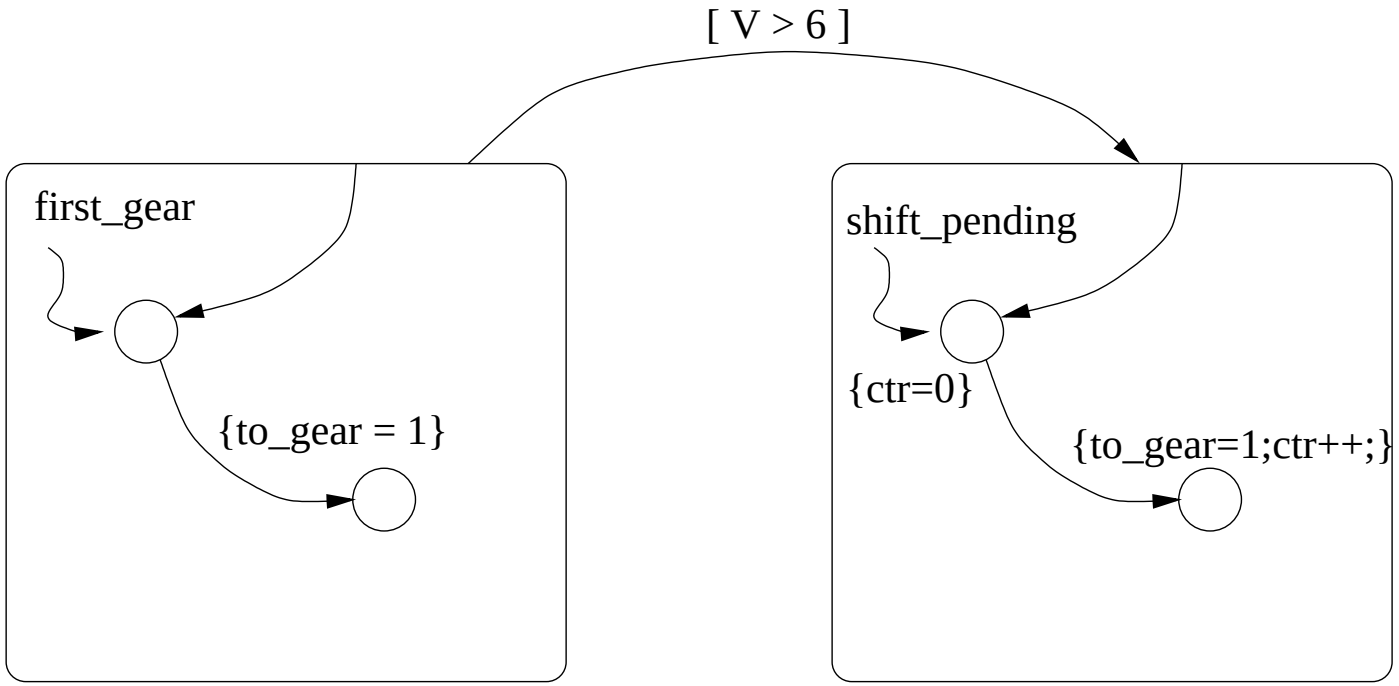
Challenge: To show that within some sampled-data constraints, the controller states and the physical plant states are consistent.

Insight: State consistency is essentially a property about the Stateflow components and we can use some abstraction of the environment.

Powertrain Control

- Stateflow components in the physical plant *and* the controller are both translated into guarded transition systems using the developed semantics.
- All predicates in the guards of the Stateflow transitions are nondeterministically set by the environment. More refined predicate abstraction of the environment (over the Stateflow guards) can also be done.
- Various state consistency properties are then model checked.

Details: Stateflow Translation



$$state == first\_gear \wedge V > 6 \quad \longrightarrow \quad state = shift\_pending;$$
$$ctr = 0;$$

Details: Stateflow Translation

```
transmission:CONTEXT =
BEGIN
  SchedulerState:  TYPE = { inactive, t55, ..., t3, t4 };
  ControllerState:  TYPE = { inact, first, second, shift12,
                             change, reset, inertia12 };
  TransmissionState:  TYPE = { tfirst, torque12, tinertia12,
                               tsecond, not_engaged };
  Turn:  TYPE = {sch, con, tra, env};
```

Details: Stateflow Translation

```
system :  MODULE =
BEGIN
  LOCAL sch_state :  SchedulerState
  LOCAL tran_state :  TransmissionState
  LOCAL con_state :  ControllerState
  LOCAL Vleq6, Vleq3 :  BOOLEAN
  ...
  TRANSITION
       (con_state = shift12) AND NOT to_gear AND NOT R1wt_gt_wcr -->
                  con_state' = reset;
                  pc' = tra;
                  tc2zero' = FALSE
```

Details: Environment Representation

boolean variable $Vleq6$: represents the predicate $V \leq 6$

boolean variable $Vleq3$: represents the predicate $V \leq 3$

Environment transitions:

$$
\begin{array}{lcl}
Vleq6 = TRUE & \longrightarrow & Vleq3' = TRUE; \\
 & & Vleq6' = TRUE \\
\Box & & \\
Vleq3 = FALSE & \longrightarrow & Vleq3' = FALSE; \\
 & & Vleq6' = FALSE \\
\Box & & \\
TRUE & \longrightarrow & Vleq3' = FALSE; \\
 & & Vleq6' = TRUE
\end{array}
$$

Details: Properties

- if control and transmission enter state “second-gear”, then they remain in that state forever.
- if controller is in “change-of-mind” state, then transmission can not go to “second-gear” without first going to “first-gear”.

$$\Box((con\_state = change) \rightarrow \Box((tran\_state \neq tsecond)\ \mathbf{W}\ (tran\_state = tfirst)))$$

$$\Box((con\_state = change) \rightarrow \Box((tran\_state \neq tsecond)\ \mathbf{U}\ (tran\_state = tfirst)))$$

Progress

- Semantics for Stateflow in SAL
- Stateflow model analysis
- Simulink to SAL
- Analysis for Simulink models

Simulink to SAL

How to reason about the environment?

General analytical methods are difficult to come up with.

$$
\begin{aligned}
\dot{V} &= \frac{1}{1644}(7920 - 25800\frac{V}{\omega_{wf}} - 0.45V^2) \\
\dot{\omega_{wf}} &= \frac{1}{2.8}(T_s - 2505 + 8000\frac{V}{\omega_{wf}}) \\
\dot{T_s} &= 2374\omega_{cr} - 6742\omega_{wf} \\
&\ldots
\end{aligned}
$$

Therefore, we reason about some discretization of the system.

Discretization

$$
\begin{aligned}
V' &= V + \frac{1}{1644}\delta(7920 - 25800\frac{V}{\omega_{wf}} - 0.45V^2) \\
{\omega_{wf}}' &= \omega_{wf} + \frac{1}{2.8}\delta(T_s - 2505 + 8000\frac{V}{\omega_{wf}}) \\
{T_s}' &= T_s + \delta(2374\omega_{cr} - 6742\omega_{wf}) \\
&\ldots
\end{aligned}
$$

Simulink does the same too.

Progress

- Semantics for Stateflow in SAL
- Stateflow model analysis
- Simulink to SAL
- Analysis for Simulink models

Analysis Technique

Symbolic simulation with *widening* for initial states given by symbolic (polynomial) expressions over the state variables.

Old state:

$$\phi(\bar{V}, \bar{W})$$

Transition:

$$\psi(\bar{V}, \bar{W}) \quad \longrightarrow \quad V = \bar{V} + (a\bar{W} + b\bar{V} + c\bar{V}^2\bar{W})\frac{1}{\bar{W}};\ W = \ldots$$

New state:

$$\exists(\bar{V}, \bar{W}).(\phi \wedge \psi \wedge V = \bar{V} + (a\bar{W} + b\bar{V} + c\bar{V}^2\bar{W})\frac{1}{\bar{W}} \wedge W = \ldots)$$

Quantifier Elimination

Using QE for real closed fields, we can solve:

$$\exists(\bar{V}, \bar{W}).(\phi \wedge \psi \wedge \bar{W}V = \bar{W}\bar{V} + (a\bar{W} + b\bar{V} + c\bar{V}^2\bar{W}) \wedge W = \ldots)$$

If we are given intervals for all state variables, then the highest degree is bounded by the degrees of polynomials in the Simulink model.

Illustration: Collision Avoidance

Consider a highly simplified collision avoidance control

$$\begin{aligned} \dot{gap} &= u - v \\ \dot{u} &= A \\ \dot{v} &= A + gap - 5 \\ A &: Input\ \ Variable \end{aligned}$$

Initial conditions (which trigger this control):

$$u - v = 0 \wedge v \geq 0 \wedge gap \geq 9$$

To prove that gap is always positive.

Illustration: Collision Avoidance

A symbolic simulation using qepcad for quantifier elimination yields the following:

$$
\begin{aligned}
\phi_0 \quad &: \quad u - v = 0 \wedge v \geq 0 \wedge gap \geq 9 \\
\phi_1 \quad &: \quad \exists(\bar{a}, g\bar{a}p, \bar{v}, \bar{u}) : \\
&\qquad gap = g\bar{a}p + \tfrac{1}{10}(\bar{u} - \bar{v}) \wedge \\
&\qquad v = \bar{v} + \tfrac{1}{10}(\bar{a} + g\bar{a}p - 5) \wedge \\
&\qquad u = \bar{u} + \tfrac{1}{10}(\bar{a}) \wedge \\
&\qquad \bar{u} - \bar{v} = 0 \wedge \bar{v} \geq 4 \wedge g\bar{a}p \geq 9 \\
&: \quad 10u - 10v + gap - 5 = 0 \wedge 10u - 10v - 99gap + 895 \leq 0 \\
&: \quad 10u - 10v + gap - 5 = 0 \wedge gap \geq 9
\end{aligned}
$$

Illustration: Collision Avoidance

Second iteration:

$$
\begin{aligned}
\phi_2 \quad : \quad & \exists(\bar{a}, g\bar{a}p, \bar{v}, \bar{u}) : \\
& \quad gap = g\bar{a}p + \tfrac{1}{10}(\bar{u} - \bar{v}) \wedge \\
& \quad v = \bar{v} + \tfrac{1}{10}(\bar{a} + g\bar{a}p - 5) \wedge \\
& \quad u = \bar{u} + \tfrac{1}{10}(\bar{a}) \wedge \\
& \quad 10\bar{u} - 10\bar{v} + g\bar{a}p - 5 = 0 \wedge g\bar{a}p \geq 9 \\
: \quad & 5u - 5v - 50gap + 452 \leq 0 \wedge 10u - 10v + gap - 5 = 0 \\
: \quad & gap \geq 9 \wedge 10u - 10v + gap - 5 = 0
\end{aligned}
$$

In general, require simplification and widening for termination.

Plan

Plan I

Analysis and verification:

- Integrate with the Vanderbilt parser
- Use techniques on larger examples made available by OEPs
- Typechecking Stateflow/Simulink models

Invisible Methods

In general, can not certify without complete understanding (unless simple classes of decidable system)

But, some automatic analysis is still possible, which can then be used further for full verification.

Checking is simpler than proving, so with some help from control designer, can (formally prove) derive more information about models.

Technology: Typechecking

Typechecking

A guarded transition

$$\phi \longrightarrow x' = \frac{1}{y} + \ldots$$

generates a type correctness proof obligation

$$\Box(\phi \Rightarrow y \neq 0)$$

We can go beyond and use the richer type system supported by PVS: specify types using Simulink blocks in the model.

For example, assertions like $x > y$, etc.

Plan II

Multiple-view modeling:

- Semantics of a time-triggered specification language in SAL
- Notion of consistency between different levels

Plan III

Test vector generation — explore different coverage criteria such as that based on signs of polynomials that appear in the description, etc.