Formal Methods for Analysis of Hybrid Systems

Ashish Tiwari

Tiwari@csl.sri.com

Computer Science Laboratory

SRI International

Menlo Park CA 94025

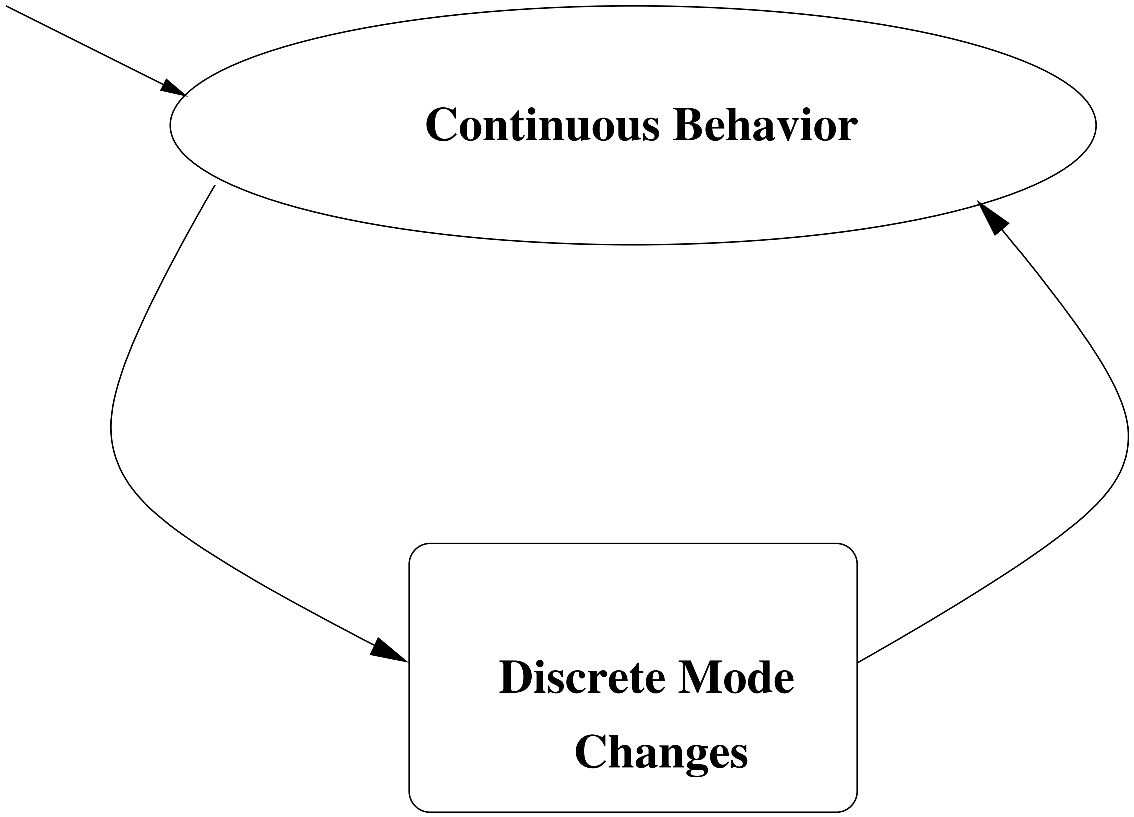
Joint work with the Formal methods group at CSL, SRI International

Analysis of Systems/Designs

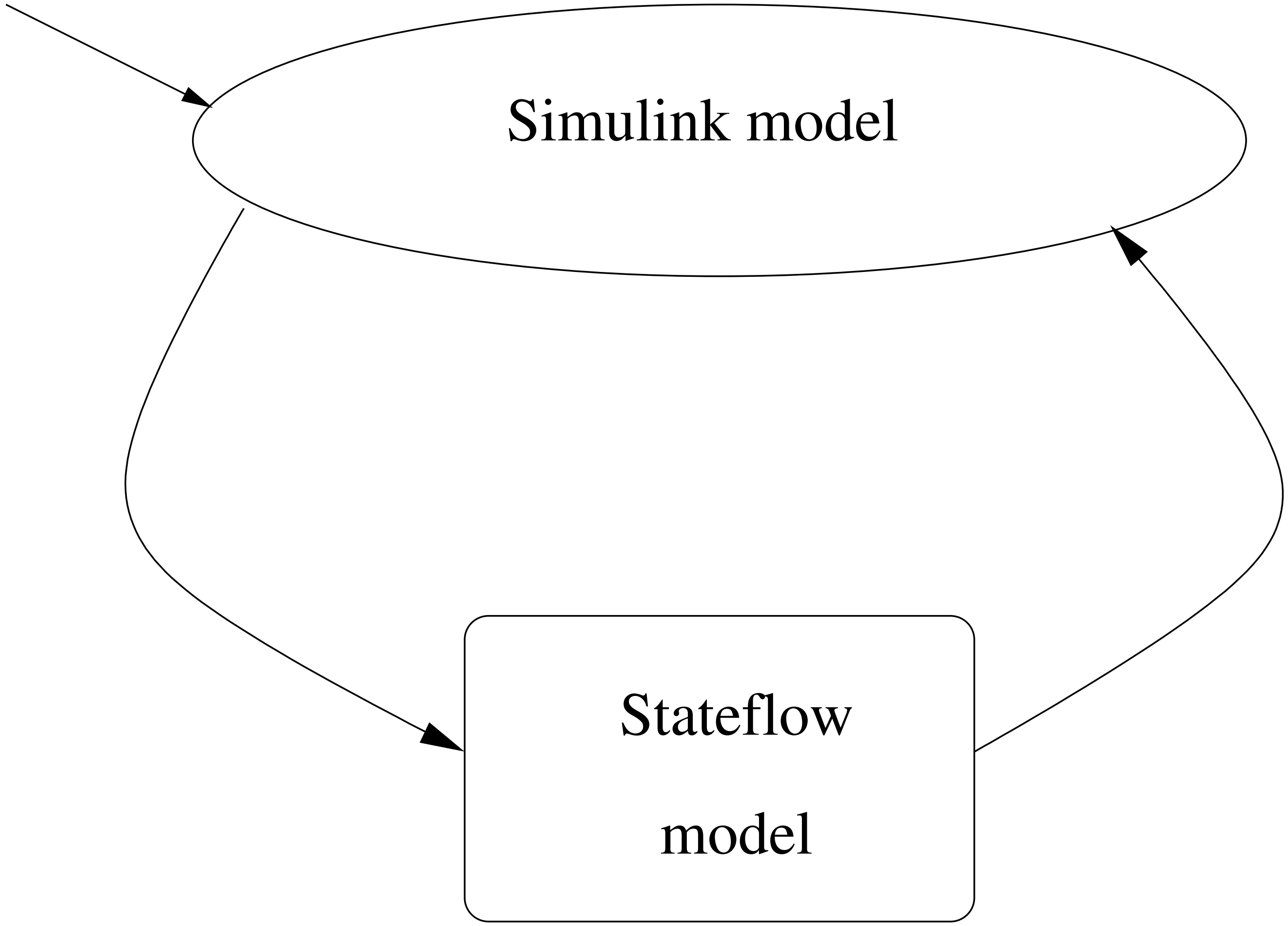
- Simulation based: Testing
  - How to generate good test vectors?
- Algorithmic: Based on certain theoretical results
  - Strong enough decidability results?
- Symbolic and semi-automatic: Using a theorem proving system
  - How much automation achievable?

Hybrid Systems

Combine discrete and continuous dynamics.



Matlab Simulink/Stateflow

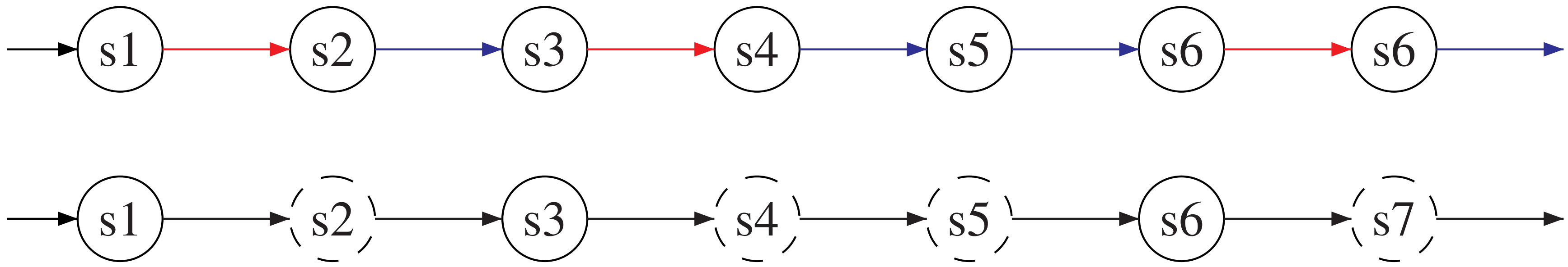


Hybrid Automata

A tuple $(Q, X, \mathbf{S}_0, F, Inv, R)$:

- Q: finite set of discrete variables
- X: finite set of continuous variables
- $\mathbf{X} = \mathbb{R}^{|X|}$, $\mathbf{Q}$ = set of all valuations for Q
- $\mathbf{S} = \mathbf{Q} \times \mathbf{X}$
- $\mathbf{S}_0 \subseteq \mathbf{S}$ is the set of initial states
- $F : \mathbf{S} \mapsto \mathbb{R}^{|X|}$ specifies the flow
- $Inv : \mathbf{Q} \mapsto 2^{\mathbb{R}^{|X|}}$ gives the invariant set
- $R \subseteq \mathbf{S} \times \mathbf{S}$ captures discontinuous state changes

Semantics of Hybrid Systems



- $s1 \in \mathbf{S}_0$ is an initial state
- Discrete Evolution: $s_i \rightarrow s_{i+1}$ iff $R(s_i, s_{i+1})$
- Continuous Evolution: $s_i = (l, x_i) \rightarrow s_{i+1} = (l, x_{i+1})$ iff there exists a $f : \mathbb{R}^{|X|} \mapsto \mathbb{R}^{|X|}$ and $\delta > 0$ such that

$$\begin{array}{rclcrcl} x_{i+1} & = & f(\delta) & \quad & x_i & = & f(0) \\ \dot{f} & = & F(l) & \quad & f(t) & \in & Inv(l) for 0 \leq t \leq \delta \end{array}$$

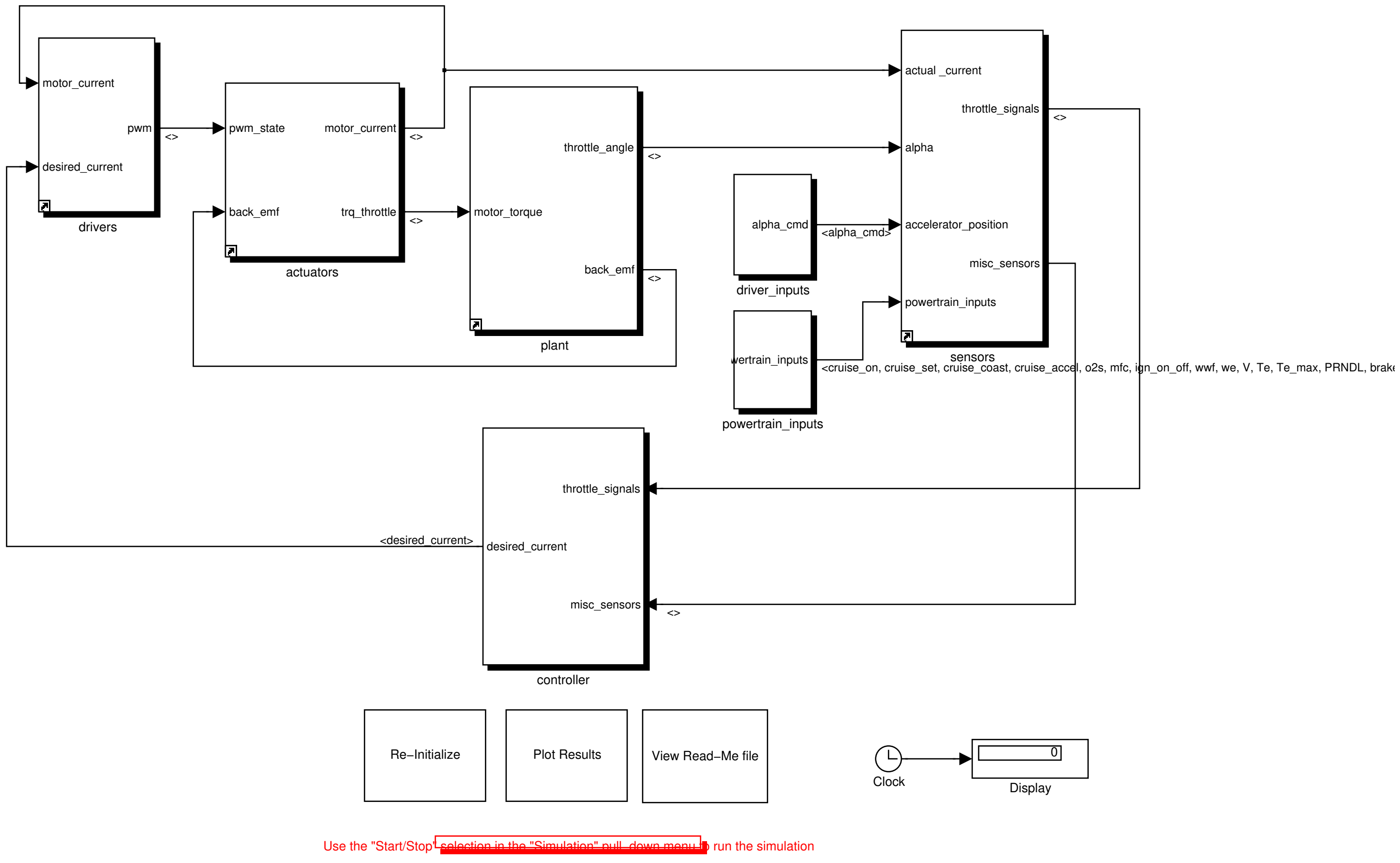
Theoretical Results

Reachability (and in some cases LTL/CTL model checking) is decidable for:

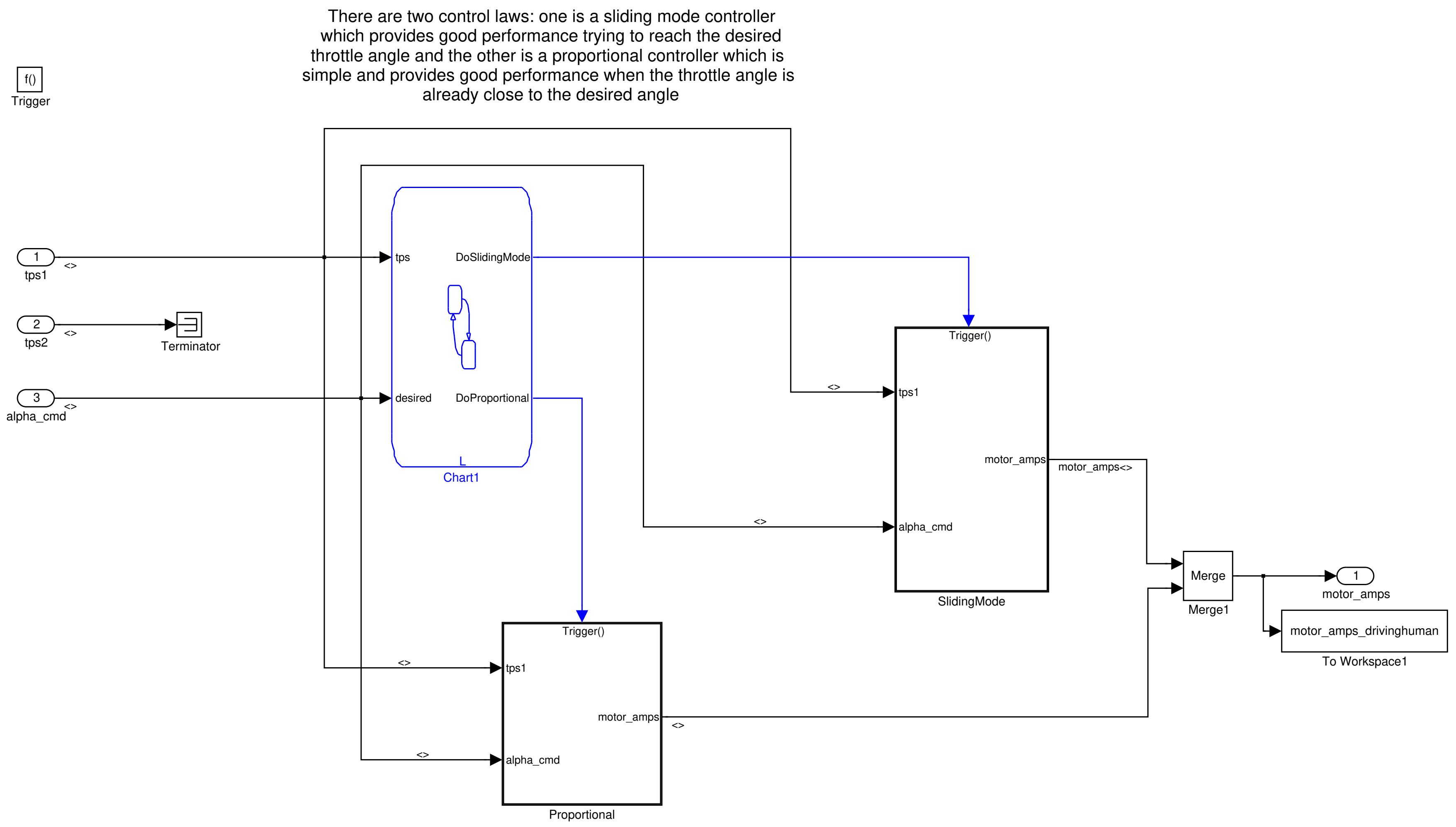
- Timed Automata: Only clocks allowed as continuous variables.
- Initialized Multirate Automata: $\dot{x} = c$ for all continuous variables x and discrete transitions reset variables changing rates.
- Initialized Rectangular Automata: $\dot{x} \in [a, b]$ and discrete transitions reset variables changing rates.
- O-minimal automata with flow specified by linear differential equation and discrete transitions by semialgebraic sets + some more restrictions.

Analysis Challenges

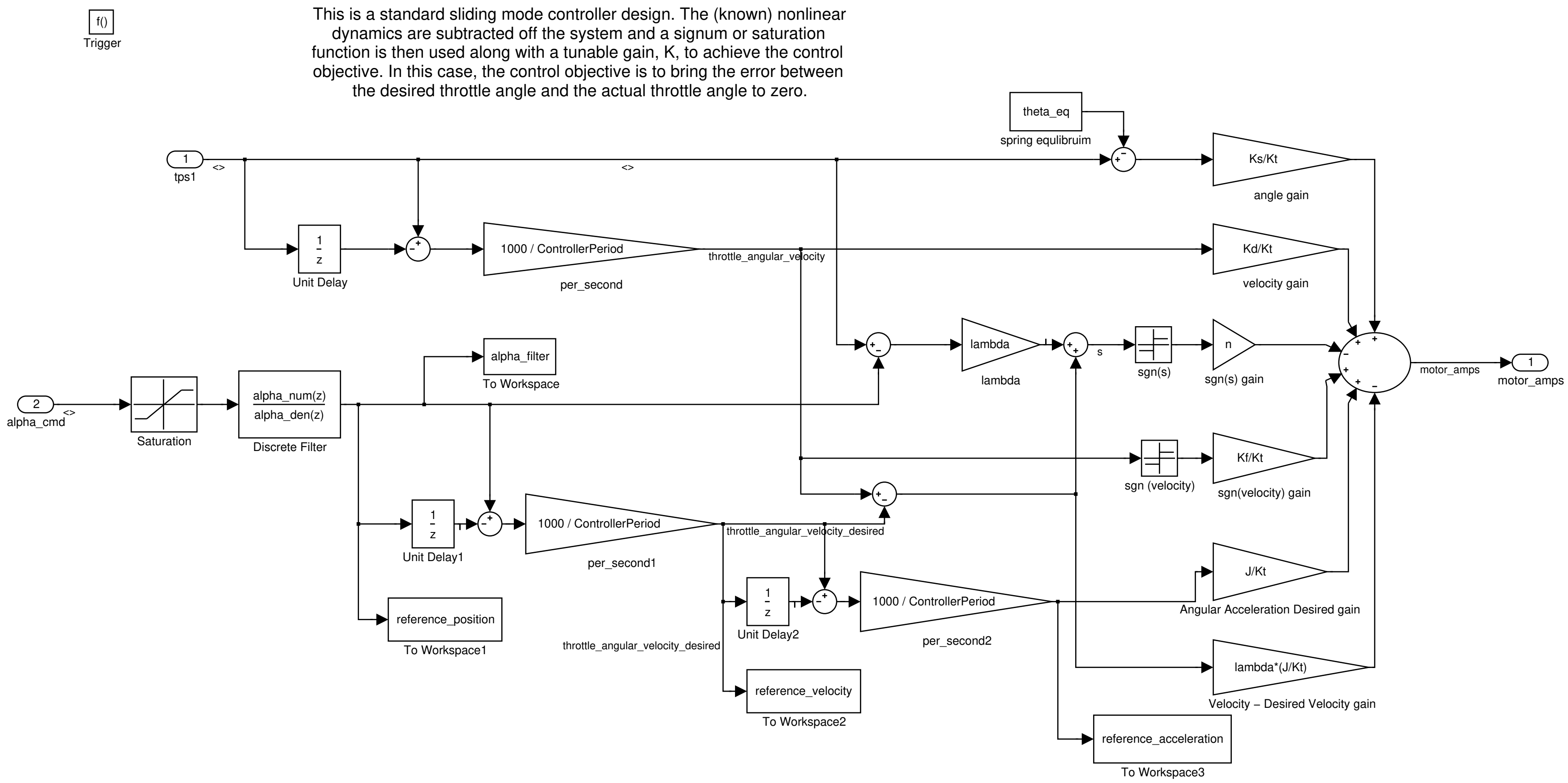
Electronic Throttle Control (ETC) Example
Developed by Vehicle Dynamics Lab
Paul Griffiths, Jason Souder, Mark Wilcutts
University of California, Berkeley
Version 2.0



Somewhere deep inside...

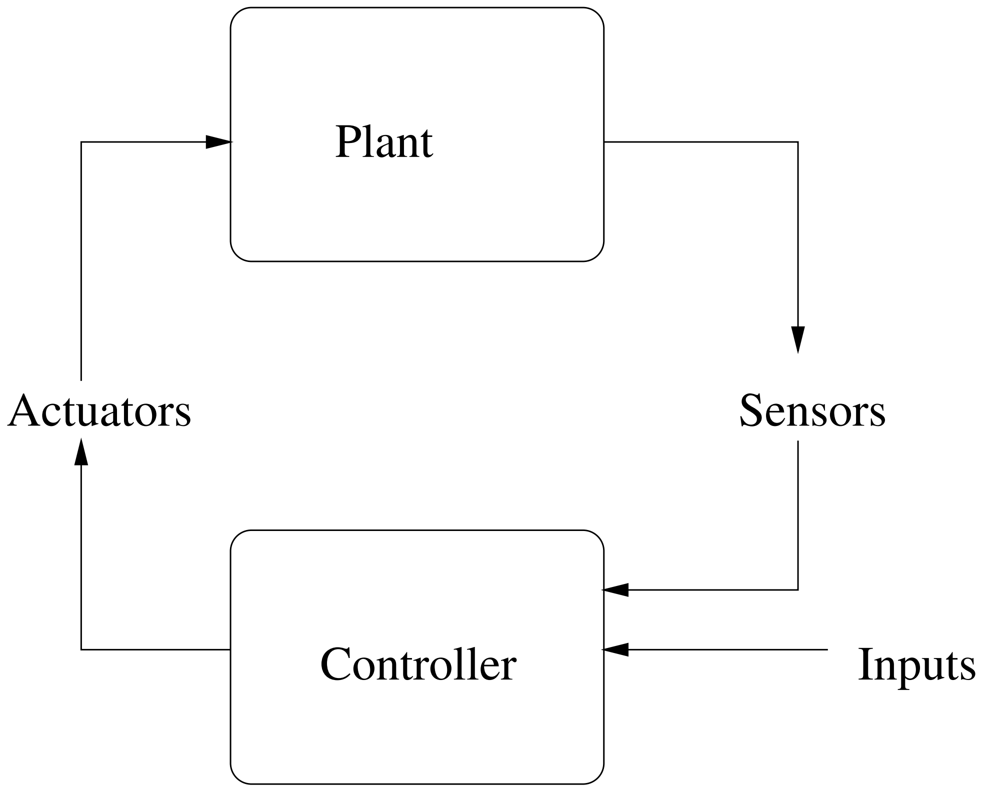


And finally...


f()
Trigger
This is a standard sliding mode controller design. The (known) nonlinear dynamics are subtracted off the system and a signum or saturation function is then used along with a tunable gain, K, to achieve the control objective. In this case, the control objective is to bring the error between the desired throttle angle and the actual throttle angle to zero.
theta_eq
spring equlibruim
1
tps1
Ks/Kt
angle gain
1
z
Unit Delay
1000 / ControllerPeriod
per_second
throttle_angular_velocity
Kd/Kt
velocity gain
lambda
lambda
s
sgn(s)
n
sgn(s) gain
motor_amps
1
motor_amps
alpha_filter
To Workspace
2
alpha_cmd
Saturation
alpha_num(z)
alpha_den(z)
Discrete Filter
sgn (velocity)
Kf/Kt
sgn(velocity) gain
Unit Delay1
1000 / ControllerPeriod
per_second1
throttle_angular_velocity_desired
Unit Delay2
1000 / ControllerPeriod
per_second2
J/Kt
Angular Acceleration Desired gain
reference_position
To Workspace1
throttle_angular_velocity_desired
reference_velocity
To Workspace2
lambda*(J/Kt)
Velocity – Desired Velocity gain
reference_acceleration
To Workspace3

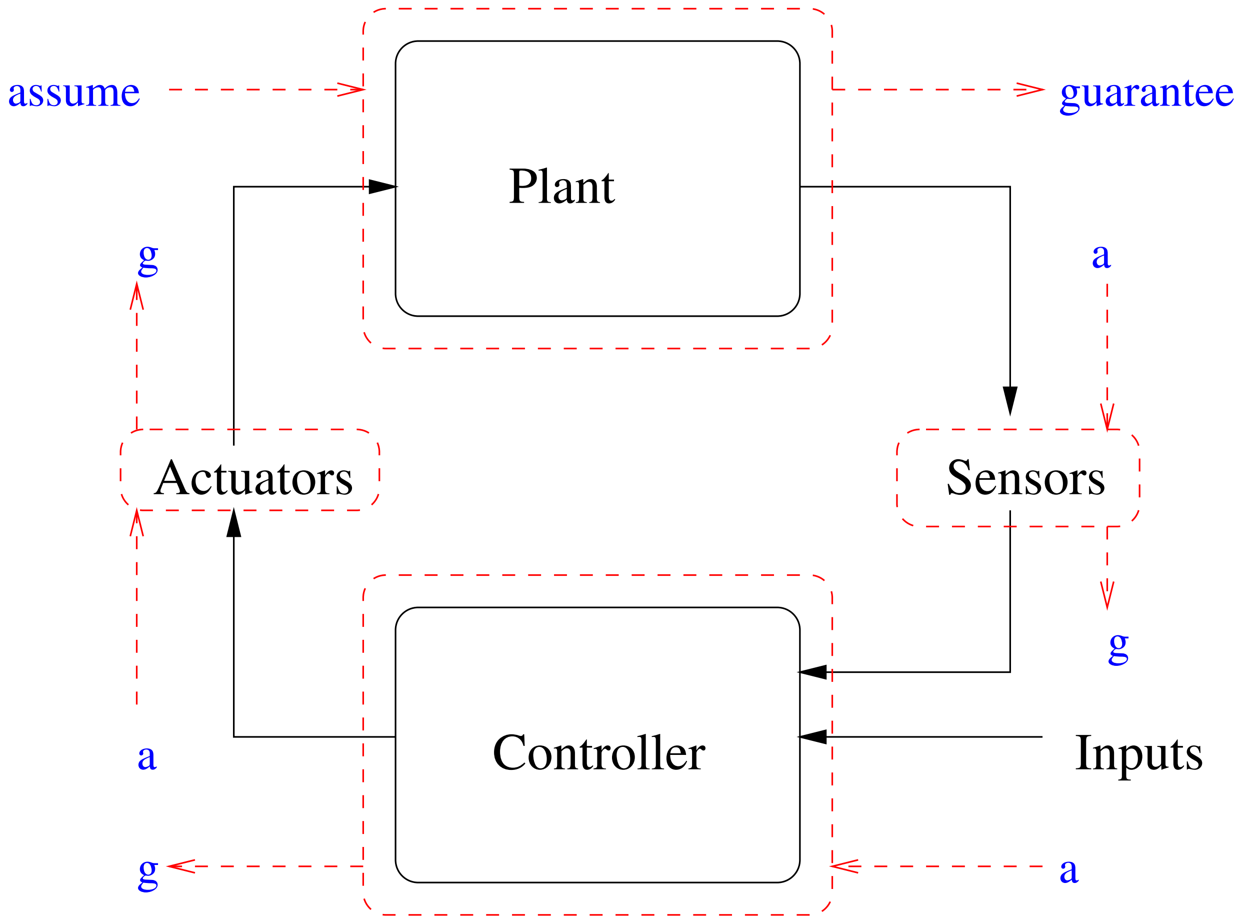
Typical Control System

Multiple levels of detail of the following:



- Controller: Contains time triggered subsystems
- Plant: Purely continuous

Modular Analysis



How do I...

- check an assume-guarantee pair of assertions?
- generate a "guarantee" given an "assume" for a module?

And these in the context of hybrid systems.

Furthermore, can I get an incremental methodology: successively stronger "assumptions" yield successively stronger "guarantees", perhaps using more and more effort?

Simple Typechecking

If the initial values of all the input and state variables are of some specified types,
then the final values of the output and state variables are of the specified types.

More information in types:

- Simple types: ints, reals, boolean
- Predicate subtypes: subrange, nonzero reals, positives, even, $\{x : \phi(x)\}$
- Dependent types: whenever x is even, y is odd, $\{y : y > f(x)\}$

But, how to do typechecking in presence of continuous transitions?

Invariant Checking for Hybrid Systems

Problem: Discrete transitions are OK, but for continuous transitions...?

Continuous Evolution:

$s_i = (l, x_i) \rightarrow s_{i+1} = (l, x_{i+1})$ iff there exists a $f : \mathbb{R}^{|X|} \mapsto \mathbb{R}^{|X|}$ and $\delta > 0$ such that

$$\begin{array}{rclcrcl} x_{i+1} & = & f(\delta) & \quad & x_i & = & f(0) \\ \dot{f} & = & F(l) & \quad & f(t) & \in & Inv(l) for 0 \leq t \leq \delta \end{array}$$

...need a solved form f.

Fixed Step Discretizations

Replace continuous differential equation

$$\dot{\vec{x}} = F(l)(\vec{x})$$

by difference equation

$$next(\vec{x}) = \vec{x} + c.F(l)(\vec{x})$$

- Do not need to solve the differential equations
- Under some assumptions on the flow F and discrete transition relation R, we can now do symbolic propagation and invariant checking
- Successfully did some simple examples: thermostat, adaptive cruise control

Next State Operator

If the formula $\phi(Q, X)$, over discrete variables Q and continuous variables X, represents a set of states, then the formula

- $\exists(Q_0, X_0) : \phi(Q_0, X_0) \;\wedge\; R((Q_0, X_0), (Q, X))$: specifies the next states reached by a discrete transition, and
- $\exists(Q_0, X_0) : (\phi(Q_0, X_0) \;\wedge\; X = X_0 + c.F(Q_0)(X_0) \;\wedge\; Q = Q_0$ $\wedge\; (\forall d, Y : 0 \leq d \leq c \wedge Y = X_0 + d.F(Q_0)(X_0) \Rightarrow Inv(Q_0)(Y)))$ specifies the next states reached by a continuous transition

Let $Next(\phi)$ be the disjunction of these two formulas.

Typechecking or Inductive Invariant Checking

A formula ϕ is an inductive invariant if:

$$Next(\phi) \vee Init \Rightarrow \phi$$

If we assume that the discrete state space, $\mathbf{Q}$, is finite (as is usually the case), then the crucial theory for

- computing the $Next$ operator, and
- invariant checking

is that of the real closed fields.

FO Theory of $\mathbb{R}$

Language	$\{0, 1, \mathbb{Q}, +, -, ., <, =\}$
Term	E.g. $2x + 3xy - 1$
Atomic Formula	$2x + 3xy - 1 < 0$
FO Formula	Application of boolean operators $\{\vee, \wedge, \neg\}$ and quantifiers $\{\exists, \forall\}$.

Formulas in the theory $\mathbb{R}$ containing free variables $x_1, \ldots, x_n$ denote a semialgebraic subset of $\mathbb{R}^n$.

QEPCAD Decision Procedure

- CAD: Decomposing the real space $\mathbb{R}^n$ into finite partitions which are sign-invariant for a set P of polynomials in $\mathbb{R}[x_1, \ldots, x_n]$

  Example: $x^2 - 4$ creates partitions $x < -2, x = -2, -2 \leq x < 2, x = 2,$ and $2 < x$.

- QE: Get all polynomials in the formula ϕ, create the CAD decomposition, and evaluate formula on sample points.

The QEPCAD implementation is done over the SACLIB package.

Approach

- PVS: An interactive theorem prover that has been extensively used for formal verification.
- QEPCAD: A decision procedure for the first-order theory of real closed fields.
- SAL: Intermediate language for transition systems.

Integrating QEPCAD with PVS

- Black box integration: PVS submits a quantified formula in prenix-normal form to QEPCAD and QEPCAD returns a quantifier-free formula.
- Heuristics: Certain preprocessing is done to reduce the size of the formulas submitted to the QEPCAD package.
  - Separate part not containing the quantified variable:

$$\exists x : (\phi_1(x) \wedge \phi_2) \Leftrightarrow (\exists x : \phi_1(x)) \wedge \phi_2$$

  - Solve, if possible, for certain variables:

$$\exists x : \phi_1(x) \wedge (x = t) \Leftrightarrow \phi_1(x/t)$$

Symbolic Simulation

Assuming that:

- the initial set of states, $\mathbf{S}_0$, is semialgebraic
- the Flow function at each discrete location l, $F(l)$, is specified using polynomials
- the invariants, $Inv(l)$, at each discrete location l are specified in the FO theory of reals
- the discrete transition relation, R, also uses only polynomials in the specification of the guards and resets

Then, the one step $Next$ operator is computable and symbolic simulation and invariant checking are feasible.

Illustration: Collision Avoidance

Consider a highly simplified collision avoidance control

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

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:

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

Illustration: Collision Avoidance

Second iteration:

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

In general, require simplification and widening for termination.

Symbolic Step Discretization

Replace the continuous differential equation

$$\dot{\vec{x}} = F(l)(\vec{x})$$

by the difference equation

$$next(\vec{x}) = \vec{x} + \delta.F(l)(\vec{x})$$

where δ is a symbolic constant constrained to be in a certain small range $[0, c]$.

Owing to the symbolic nature of our analysis technique, invariant checking and symbolic simulation can still be carried out on this formulation.

Invariant Generation

How to generate "guarantees" from "assumptions"?

- Symbolic propagation with widening and narrowing
  - Require computability of the $Next$ state operator
  - Use widening to reach fixpoint
  - Check for fixpoint is identical to invariant checking
- Abstraction and get reachable states
  - How to abstract the continuous dynamics?
  - Successive refinements possible?

Abstraction for Hybrid Automata

Consider a simple thermostat controller with:

- Discrete modes: Two modes, $q = on$ and $q = \mathit{off}$
- Continuous variable: The temperature x
- Initial State: $q = \mathit{off}$ and $x = 75$
- Discrete Transitions:

$$
\begin{aligned}
q = \mathit{off} \ \ and \ x \leq 70 \quad &\longrightarrow \quad q = on \\
q = on \ \ and \ x \geq 80 \quad &\longrightarrow \quad q = \mathit{off}
\end{aligned}
$$

Thermostat Example Contd

- Continuous Flow:

$$
\begin{array}{rcl}
q = \mathit{off}\ \ and\ x > 68 & \longrightarrow & \dot{x} = -Kx \\
q = on\ \ and\ x < 82 & \longrightarrow & \dot{x} = K(h - x)
\end{array}
$$

Interested in proving that the temperature x is always between 68 and 82.

Abstraction Algorithm

Consists of three steps:

- Seed: Start with a set P_0 of polynomials of interest

$$P_0 = \{x - 68, x - 82\}$$

- Saturate: Add the polynomials in guards and close the set under derivatives of the polynomials

$$\begin{aligned} P_1 &= \{x - 68, x - 82, x - 70, x - 80\} \\ P_2 &= P_1 \cup \{-Kx, K(h - x)\} \\ P_3 &= P_1 \cup \{x, h - x\} \end{aligned}$$

- Abstract

Abstract System

- Abstract Variables:
  - q
  - $\{q_1, q_2, \ldots, q_{|P_3|}\}$, one corresponding to each polynomial in P_3
- Abstraction Map:

$$
\begin{array}{rclcrcl}
q_i & = & 1 & \textit{iff} & p_i & > & 0 \\
q_i & = & 0 & \textit{iff} & p_i & = & 0 \\
q_i & = & -1 & \textit{iff} & p_i & < & 0
\end{array}
$$

- Abstract State Space: $2 \times 3^{|P_3|}$

Abstracting Continuous Transitions

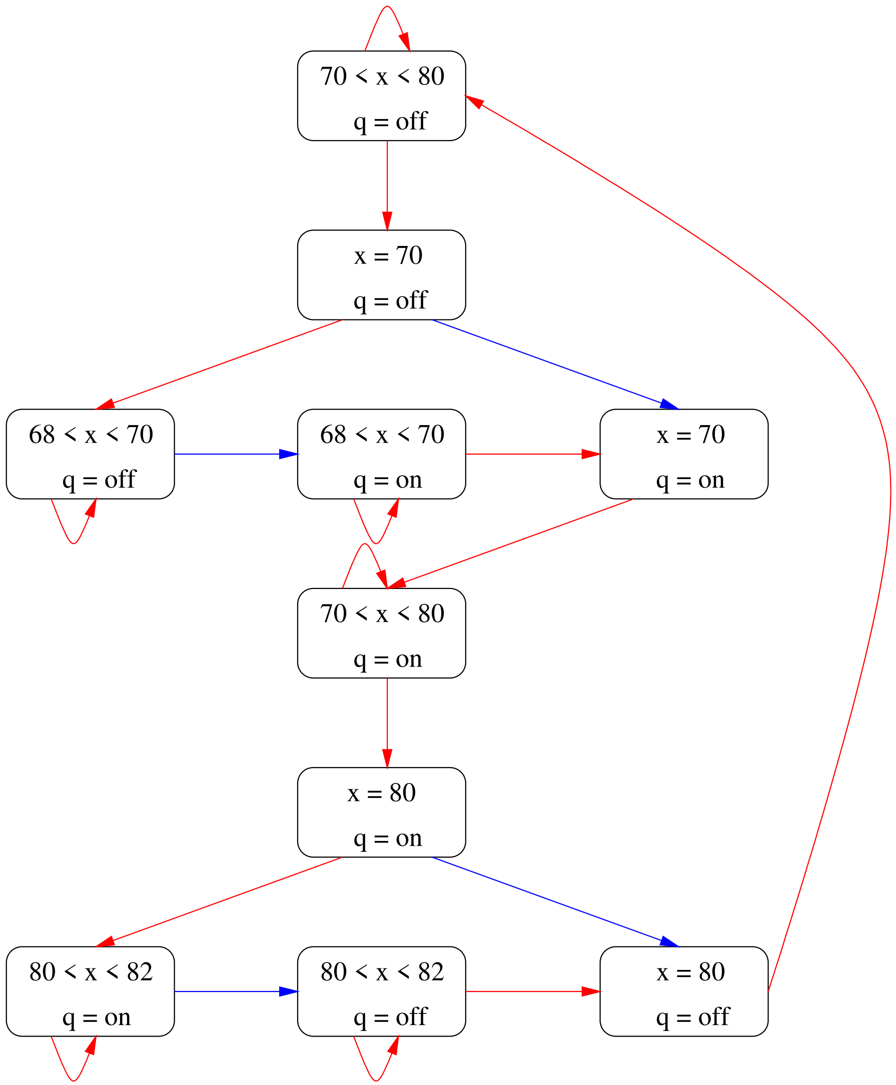
- l: discrete mode
- q_i, q_j : abstract variables
- p_i, p_j : corresponding concrete polynomials

Assume: $\dot{p_i} = p_j$ in the discrete mode l.

We apply rules of the form:

- if $q_i = 1$ and $q_j = 1$, then new value of q_i is 1.
- if $q_i = 1$ and $q_j = 0$, then new value of q_i is 1.
- if $q_i = 1$ and $q_j = -1$, then new value of q_i is either 1 or 0.

Abstract Thermostat System



Features of the Abstraction Algorithm

- Quite general in its applicability
- Simple computational steps involved
- Abstractions can be refined by adding more polynomials
- The abstractions are sound with respect to the hybrid systems semantics

Tool Suite

Tools can be iteratively applied to establish stronger properties of the system

- Simple typechecking
- Invariant Checking
- Invariant Generation using propagation, widening, and narrowing
- Abstraction

Other Approaches

- Assume simplified dynamics on small time scales and design laws for the simple case
- Formally prove correctness of the design
- Have a strategy layer above to deal with errors in assumption

Lessons

- If there is a reason why the design works, then usually there is a simple formal proof of its correctness as well
- Although QEPCAD is a decision procedure, occassionally it fails
- Integrating with Simulink/Stateflow is time consuming because of the under-specified/vague semantics of such simulation tools
- Domain knowledge and specific techniques? Numerical analysis, Dynamical systems, . . .

Summary

- Used a theorem prover, with suitably integrated decision procedure, to build automated and semi-automated tools to handle state transition systems
- Developed and implemented abstraction algorithm for hybrid systems

This is continuing work

- Further experiments
- Improve and further automate existing tools
- Develop new tools and techniques