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

Outline

1. Specification of the system
2. Specification of the property
3. Formal verification techniques

Systems

Characterized by a state space and some “behavior” over this space, i.e., sets of trajectories

- Discrete Transition Systems
  - model hardware and software
- Continuous Dynamical Systems
  - model physical world processes
- Hybrid Systems
  - model the combinations

Specification of Systems: State space

Valuations of a (usually finite) set of variables over $\mathbb{R}$, $\mathbb{N}$, $\mathbb{Z}$, $\{\mathit{True}, \mathit{False}\}$, finite enumerations types, and so on.

System	Variables range over
Discrete	countable domains
Dynamical Systems	reals
Hybrid Systems	both

Specification of Systems: Behavior

Global behaviors are specified using local behaviors.

- For discrete transitions, specify how to go from one state to the next
- For continuous evolution, specify the time derivative of each variable

Formal Language

Symbols:	Constants, Function Symbols, Predicate Symbols
	Example: $0, 1, +, -, ., <, \leq, =$
Terms:	Constructed using constants, variables, function symbols
	Example: 0, $x + 1$, $xy + x$
Atomic Formulas:	Constructed using predicates on terms
	Example: $x + 1 > 0$, $xy + x = x + 1$,
Formulas:	Logic connectives on atomic formulas
	Example: $x + 1 > 0\ AND\ xy + x = x + 1$

Representing Sets of States

Subsets of state space are represented using formulas:

$x = 72\ \ AND\ \ heaterOn = TRUE$	Single state
$x > 72\ \ AND\ \ heaterOn = TRUE$	Set of states
	$\{\langle x = 73, heaterOn = T\rangle,$
	$\langle x = 74, heaterOn = T\rangle,$
	$\langle x = 75, heaterOn = T\rangle,$
	$\vdots$

Guarded Command

Discrete transitions can be specified using a guarded command:

condition $\longrightarrow$

value of variables in new state = expression over values of variables in old state

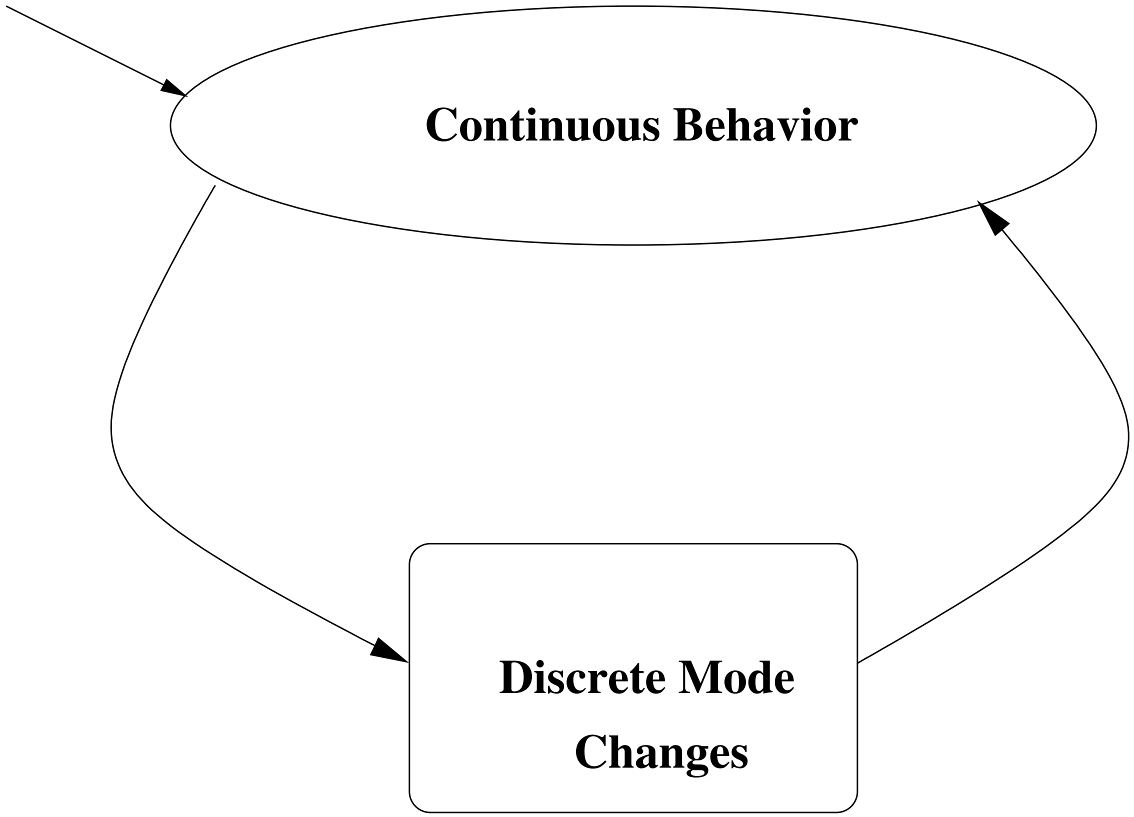
If $condition$ is true (of values of variables), then update the variables as specified on the right-hand side.

Example:

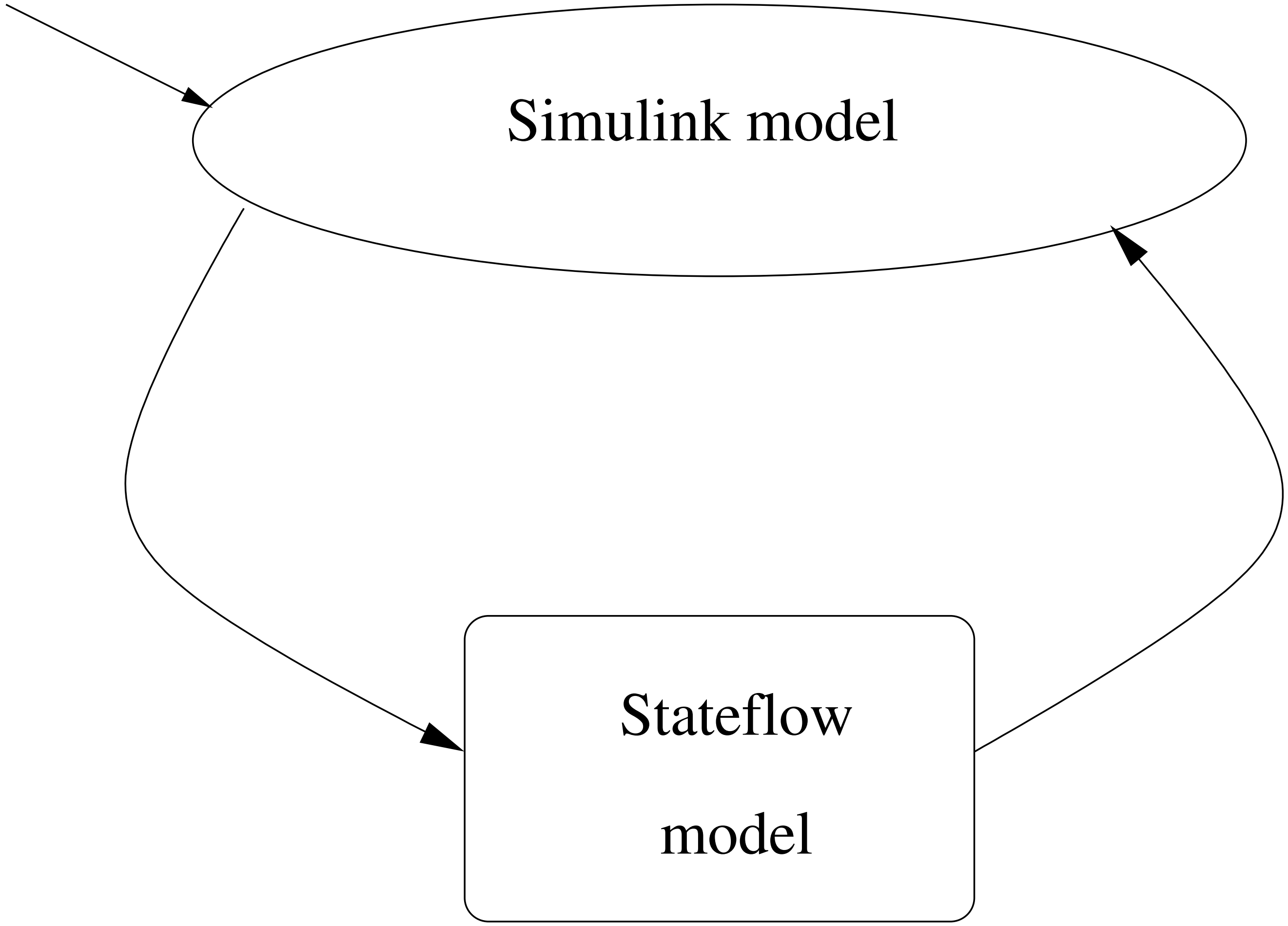
$x > 80$ AND heaterOn = TRUE $\longrightarrow$ heaterOn' = FALSE

Hybrid Systems

Combine discrete and continuous dynamics.



Matlab Simulink/Stateflow

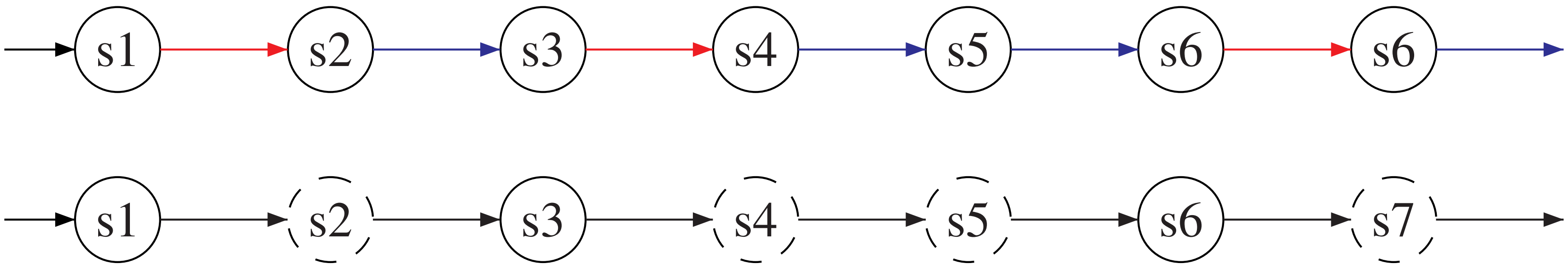


Hybrid Automata

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

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

Example: Hybrid System

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:

$$
\begin{array}{lcl}
q = \mathit{off} \ \ and \ x \leq 70 & \longrightarrow & q = on \\
q' = on \ and \ x \geq 80 & \longrightarrow & q' = \mathit{off}
\end{array}
$$

Thermostat Example Contd

- Continuous Flow:

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

Example: Thermostat

```
q :  { on, off }
x :  REAL
INITIALIZATION
  q = off; x = 75
TRANSITIONS
  [
  q = off AND x <= 70 --> q' = on
  []
  q = on AND x >= 80 --> q' = off
  []
  q = off AND x > 68 --> xdot' = -Kx
  []
  q = on AND x < 82 --> xdot' = K(h-x)
  ]
```

Outline

1. Specification of the system
2. Specification of the property
3. Formal verification techniques

Specification of Property

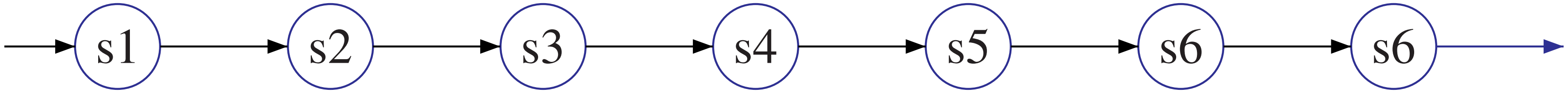
Extend the logic with always, G, and eventually, F, modalities. Further extension with path quantifiers for all paths and for some path is also possible.

For example, $G(x \geq 68\ AND\ x \leq 82)$ says that the temperature remains between 68 and 82 *always*.

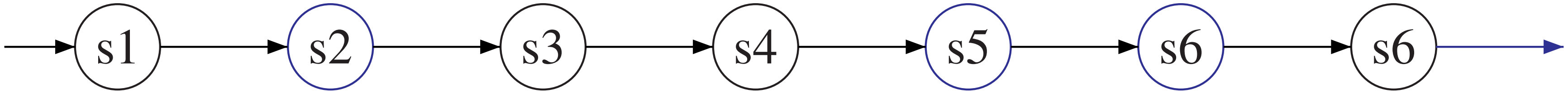
$GF(q = \mathit{off})$ says that the heater is *eventually* switched off *always*, i.e., $q = \mathit{off}$ happens infinitely often.

Properties

Property Gp holds if p is true of all states:



Property GFp holds if p is always true in some future state:



Outline

1. Specification of the system
2. Specification of the property
3. Formal verification techniques

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?

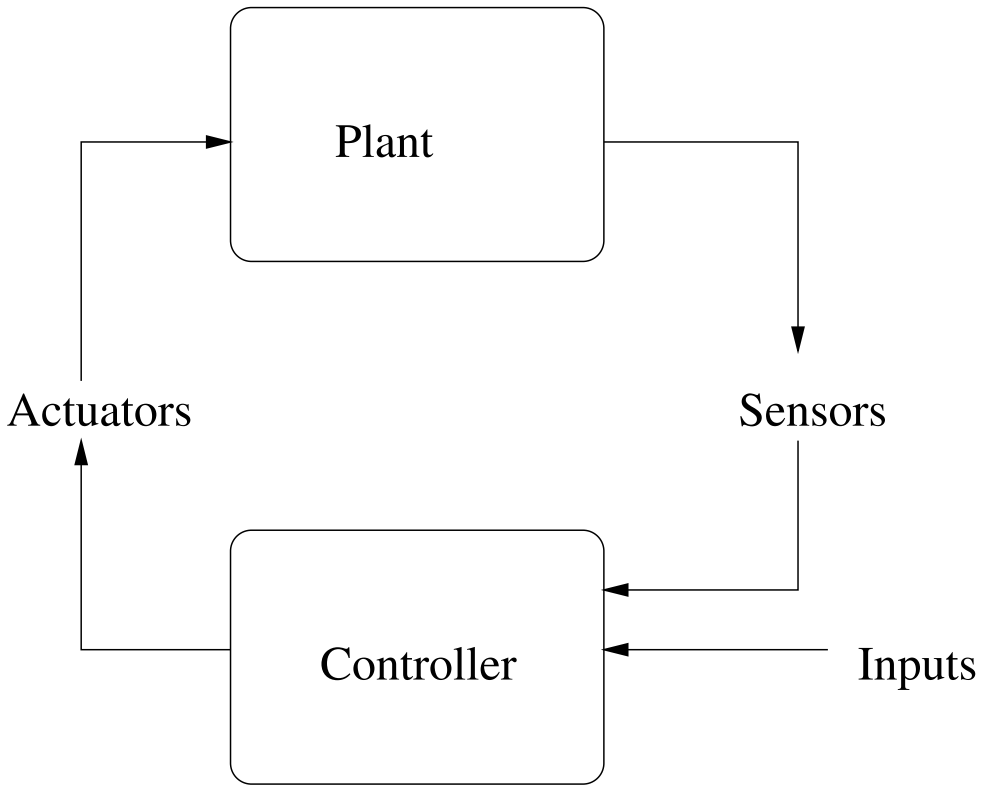
Theoretical Results for Hybrid Automata

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.

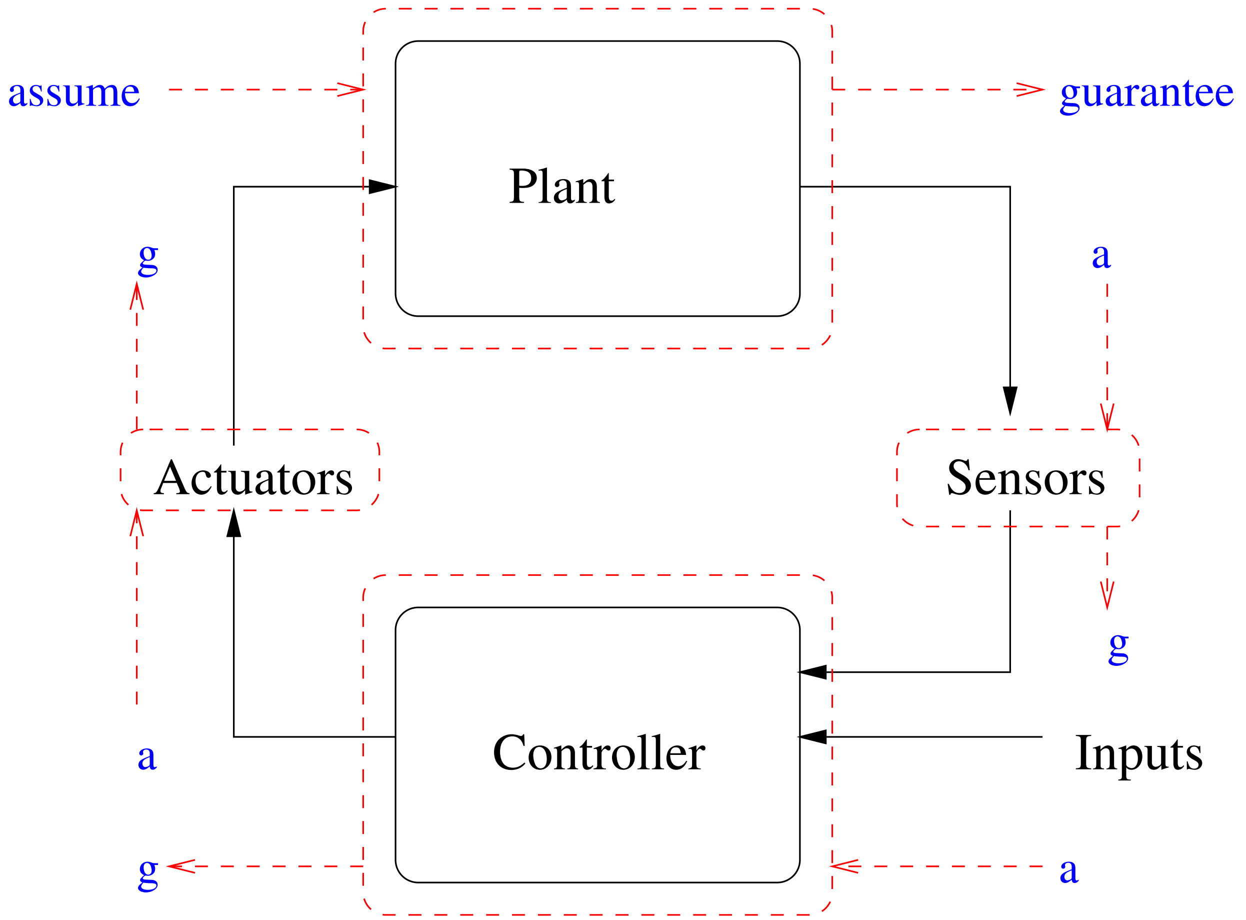
Typical Control System

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?

Typechecking or Inductive Invariant Checking

A formula ϕ is an inductive invariant (type) if:

- The property ϕ is true of the initial state
- If property ϕ is true in a state, then ϕ is also true in the state obtained after one transition.

A property ϕ may be "true always", but not inductive. For example, in collision avoidance, the property that "the cars do not collide" is not inductive. In the thermostat, the property that

$$q = \mathit{off} \Rightarrow 70 \leq x \leq 80$$

is not inductive.

Even for checking inductive invariants, we need a computable next-state operator.

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

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

...need a solved form f.

Initial Attempts

- Fixed Step Discretizations: Replace continuous differential equation

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

by difference equation

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

- Symbolic Step Discretizations: Replace continuous differential equation

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

by 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 $[0, c]$.

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

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.

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?

Example: Thermostat

```
q :  { on, off }
x :  REAL
INITIALIZATION
  q = off; x = 75
TRANSITIONS
  [
  q = off AND x <= 70 --> q' = on
  []
  q = on AND x >= 80 --> q' = off
  []
  q = off AND x > 68 --> xdot' = -Kx
  []
  q = on AND x < 82 --> xdot' = K(h-x)
  ]
```

Abstraction Algorithm

Consists of three steps:

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

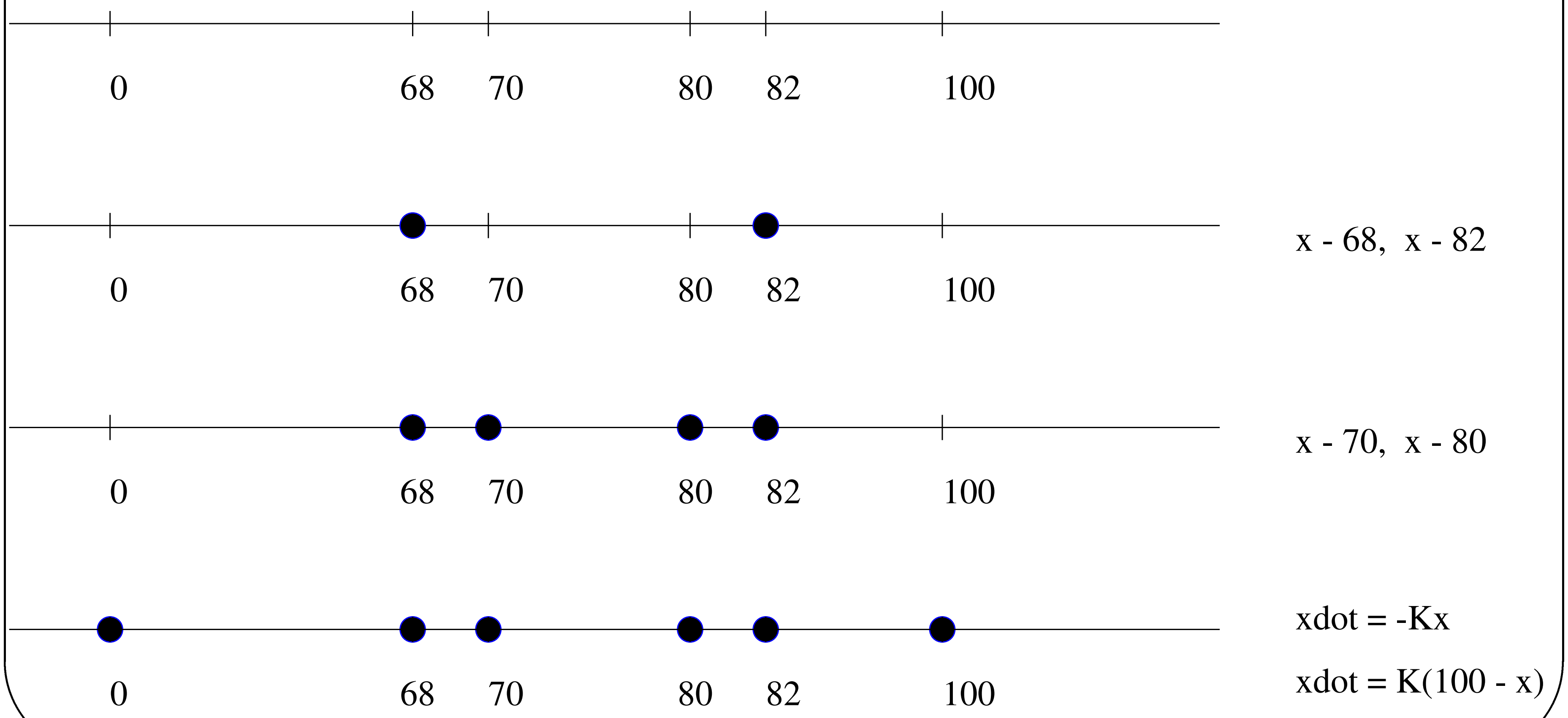
$$P_0 = \{x - 68, x - 82\}$$

- 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

Generating New Polynomials



Abstract System

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

$$\begin{array}{lllllll} q_i & = & pos & \textit{iff} & p_i & > & 0 \\ q_i & = & zero & \textit{iff} & p_i & = & 0 \\ q_i & = & neg & \textit{iff} & p_i & < & 0 \end{array}$$

- 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 $q_i = pos$ and $q_j = pos$, then new value of q_i is pos.
- if $q_i = pos$ and $q_j = zero$, then new value of q_i is pos.
- if $q_i = pos$ and $q_j = neg$, then new value of q_i is either pos or $zero$.

Removing Infeasible States

Certain abstract states are not feasible.

$$\begin{array}{lclclcl} q_1 & : & x & \qquad & q_2 & : & x - 68 \\ q_3 & : & x - 70 & & q_4 & : & x - 80 \\ q_5 & : & x - 82 & & q_6 & : & x - 100 \end{array}$$

Certain combinations like $q_2 = neg\ AND\ q_3 = pos$ are infeasible.

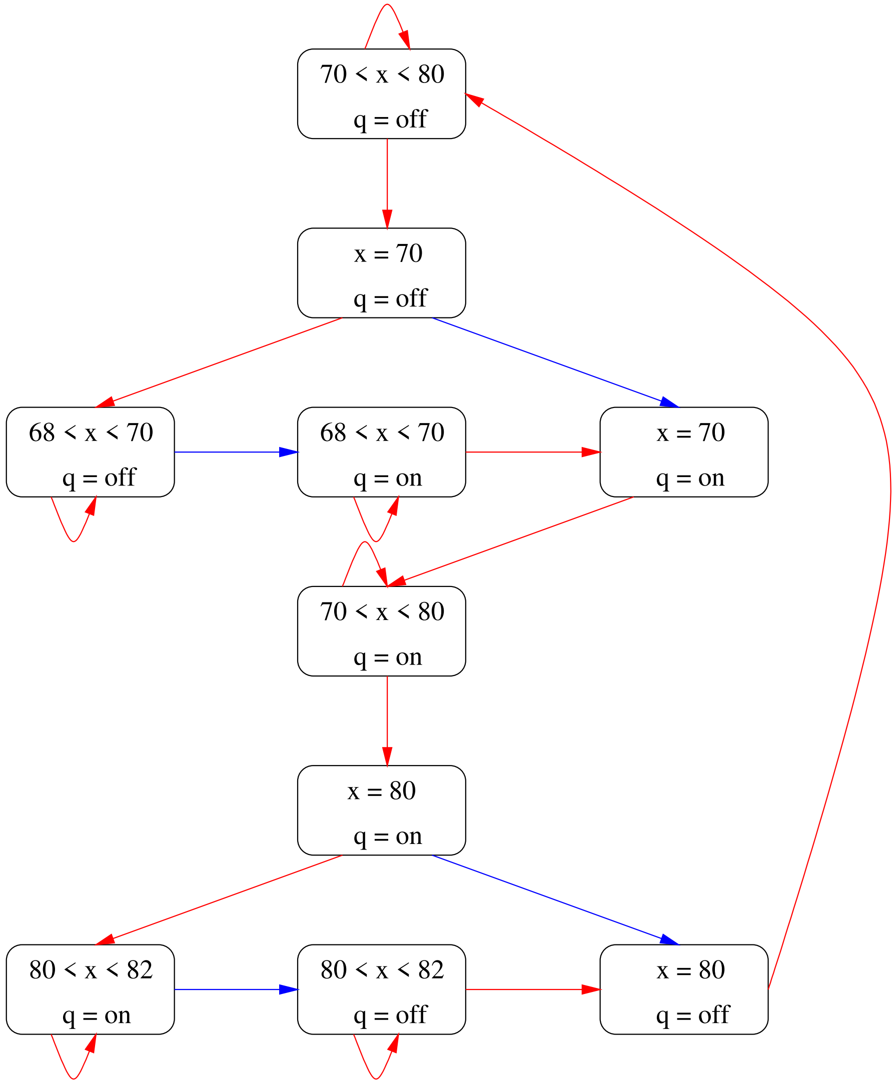
Concrete Discrete Transition:

$$q = \mathit{off}\ \ AND\ x \leq 70 \longrightarrow q' = on$$

Abstract Transition:

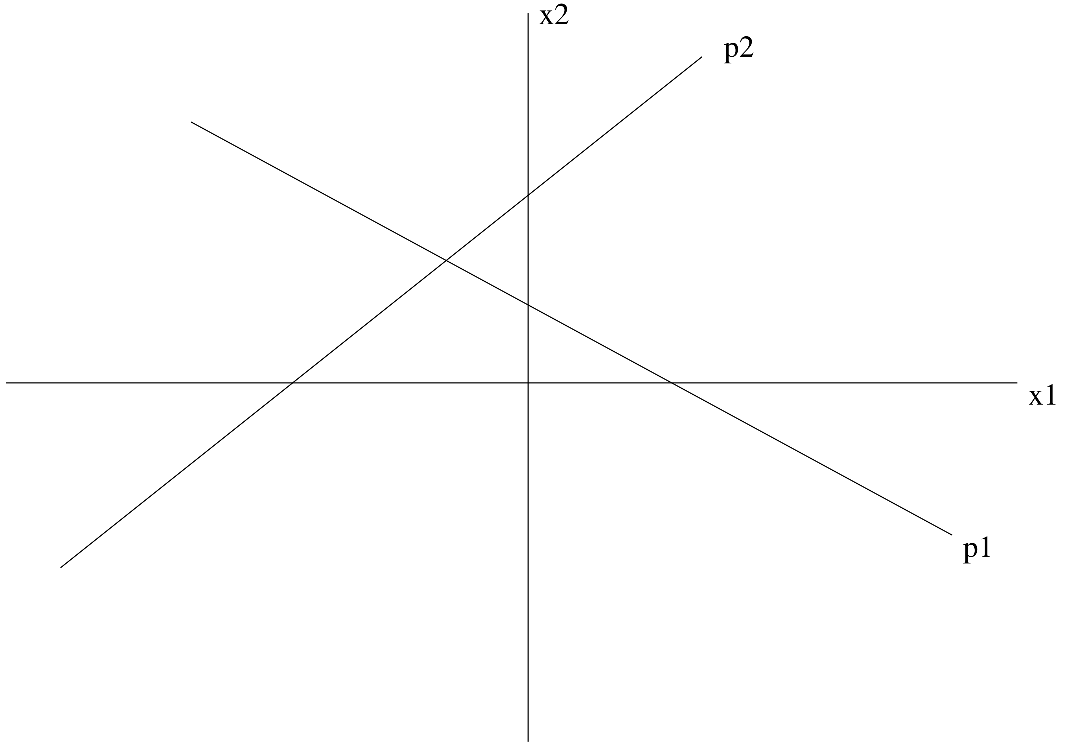
$$\begin{array}{l} q = \mathit{off}\ \ AND\ (q_3 = neg\ \ OR\ q_3 = zero)\ AND \\ \qquad q_4 = neg\ AND\ q_5 = neg\ AND\ q_6 = neg \quad \longrightarrow \quad q' = on \end{array}$$

Abstract Thermostat System



Abstraction Algorithm: 1

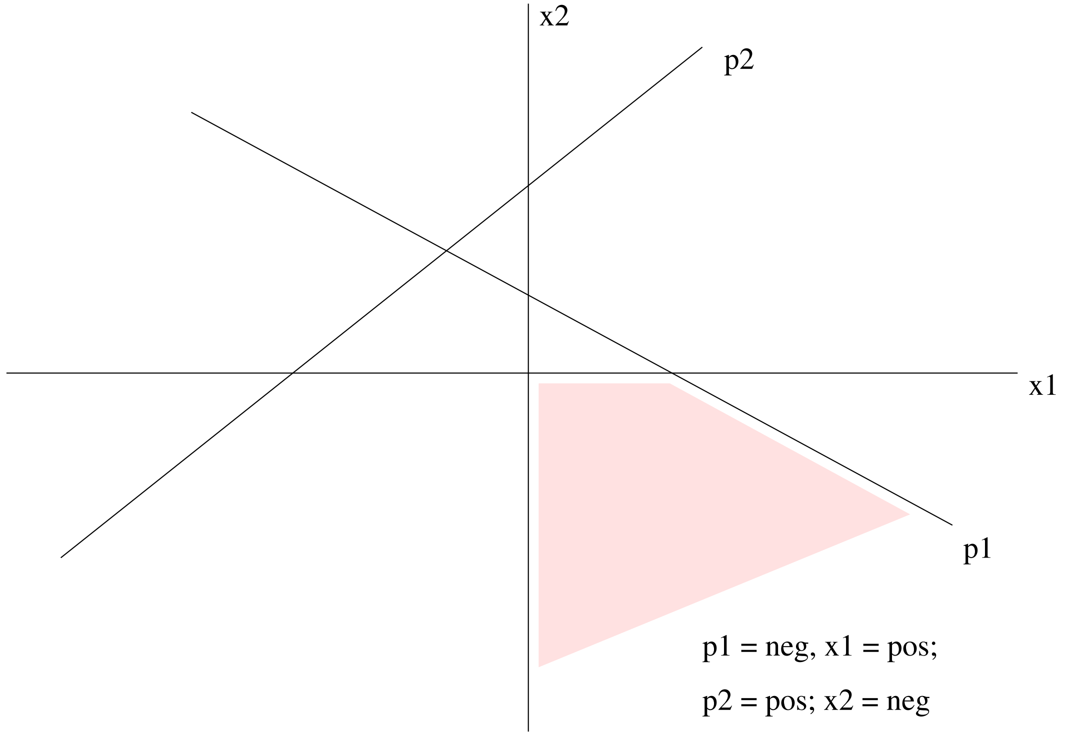
Consider a continuous dynamical system with two state variables. Concrete state space: $\Re^2$



Partitioned w.r.t signs of polynomials x_1, x_2, p_1, and p_2.

Abstraction Algorithm: 2

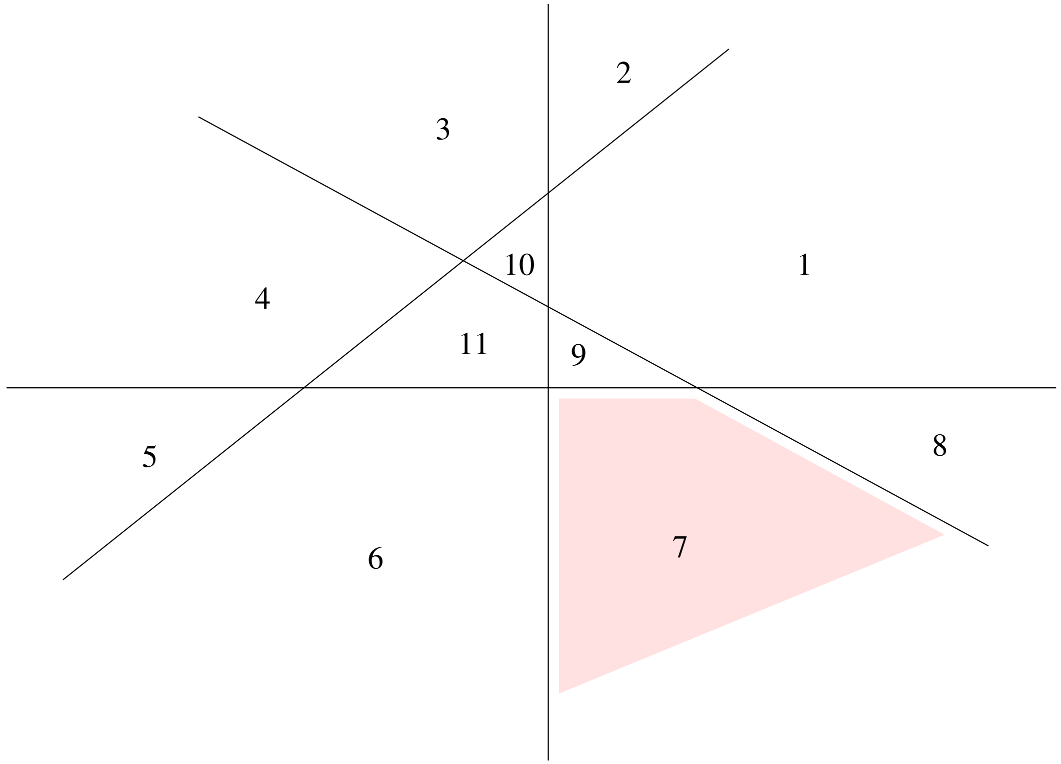
Abstract states correspond to subsets of concrete states.



More polynomials would mean more abstract states.

Abstraction Algorithm: 3

Total number of abstract states = $3^4 = 81$



Feasible abstract states = $11 + 16 + 6 = 33$

Reachable abstract states: depends on the system dynamics, possibly much less than 33.

Abstraction Algorithm: 4

- We use a mechanization of cylindrical algebraic decomposition (CAD)
- To automatically compute sound discrete abstractions to a hybrid system
- Based on the signs of the first j derivatives to the continuous variables

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

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