Hybrid Control and Switched Systems

Lecture #4
Simulation of hybrid systems

João P. Hespanha

University of California
at Santa Barbara

Summary

1. **Numerical simulation of hybrid automata**
   - simulations of ODEs
   - zero-crossing detection
2. Simulators
   - Simulink
   - Stateflow
   - SHIFT
   - Modelica

Numerical simulation of ODEs

Initial value problem (IVP) $\equiv \quad \dot{x} = f(x) \qquad x(0) = x_0$

Definition: A signal $x : [0,T] \to \mathbb{R}^n$ is a ***solution*** to the IVP if

$$x(t) = x_0 + \int_0^t f\big(x(\tau)\big)d\tau \qquad \forall t \in [0, T]$$

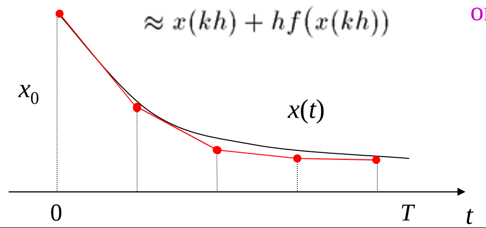
Euler method (first order method):
1st partition interval into N subintervals of length $h := T/N$

$$[kh, (k+1)h] \qquad k \in \{0, 1, \ldots, N-1\}$$

2nd assume derivative of x constant on each subinterval

$$x\big((k+1)h\big) = x(kh) + \int_{kh}^{(k+1)h} f\big(x(\tau)\big)d\tau$$
$$\approx x(kh) + hf\big(x(kh)\big)$$

on each subinterval x is assumed linear



Numerical simulation of ODEs

Initial value problem (IVP) $\equiv \quad \dot{x} = f(x) \qquad x(0) = x_0$

Definition: A signal $x : [0,T] \to \mathbb{R}^n$ is a ***solution*** to the IVP if

$$x(t) = x_0 + \int_0^t f\big(x(\tau)\big)d\tau \qquad \forall t \in [0, T]$$

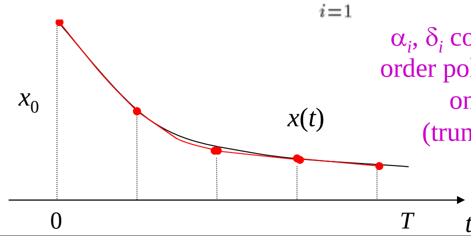
Runge-Kutta methods (*m*-order method):
1st partition interval into N subintervals of length $h := T/N$

$$[kh, (k+1)h] \qquad k \in \{0, 1, \ldots, N-1\}$$

2nd assume derivative of x constant on each subinterval

$$x\big((k+1)h\big) \approx x(kh) + h\sum_{i=1}^{m} \alpha_i f\big(x(kh) + \delta_i\big)$$

α_i, δ_i computed assuming a *m*-order polynomial approximation on each subinterval (truncated Taylor series)



Numerical simulation of ODEs

Initial value problem (IVP) $\equiv \quad \dot{x} = f(x) \qquad x(0) = x_0$

Definition: A signal $x : [0,T] \to \mathbb{R}^n$ is a ***solution*** to the IVP if

$$x(t) = x_0 + \int_0^t f\big(x(\tau)\big)d\tau \qquad \forall t \in [0, T]$$

Variable-step methods (e.g., Euler):
Pick tolerance ε and define $t_0 := 0$

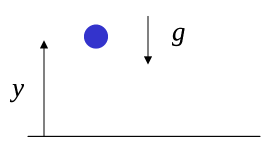
$$x\big(t_{k+1}\big) = x(t_k) + \int_{t_k}^{t_{k+1}} f\big(x(\tau)\big)d\tau$$
$$\approx x(t_k) + (t_{k+1} - t_k) f\big(x(t_k)\big)$$

choose t_{k+1} sufficiently close to t_k so that

$$\|f\big(x(t_k)\big) - f\big(x(t_{k+1})\big)\| \leq \epsilon$$

Simulation can be both *fast* and *accurate*:
1. when f is "flat" one can advance time fast,
2. when f is "steep" one advances time slowly (to retain accuracy)

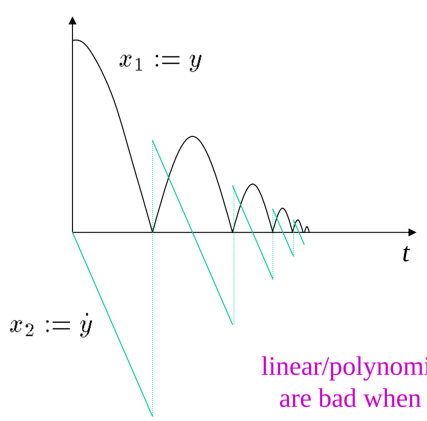
Example #1: Bouncing ball

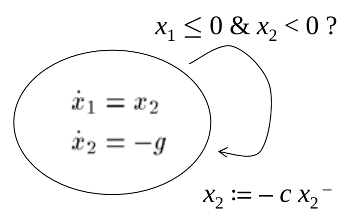


Free fall $\equiv \quad \ddot{y} = -g$

Collision $\equiv \quad y(t) = y^-(t) = 0$
$\dot{y}(t) = -c\dot{y}^-(t)$

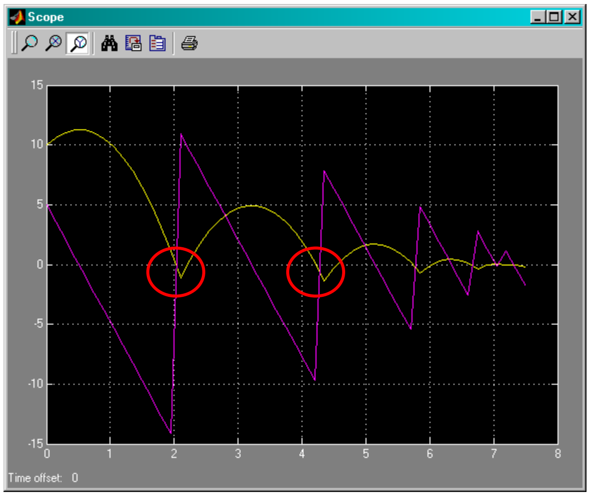
$c \in [0,1) \equiv$ energy absorbed at impact

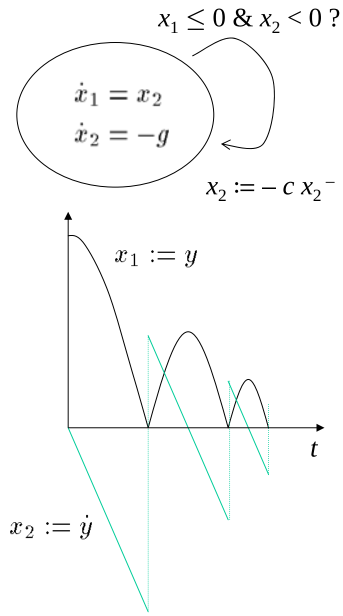




linear/polynomial approximations
are bad when transitions occur

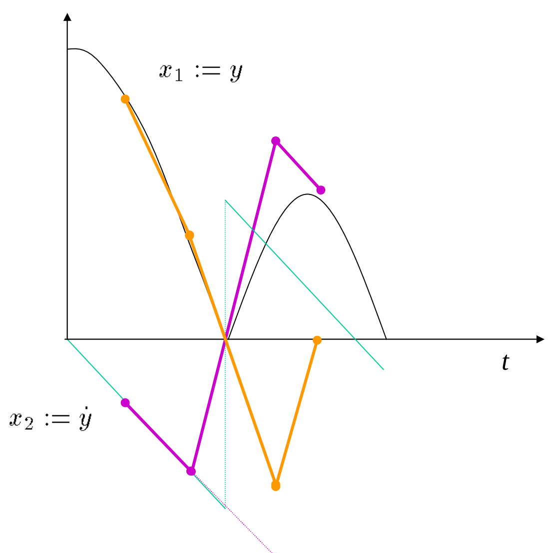
Example #1: Bouncing ball





ball_nozerocross.mdl

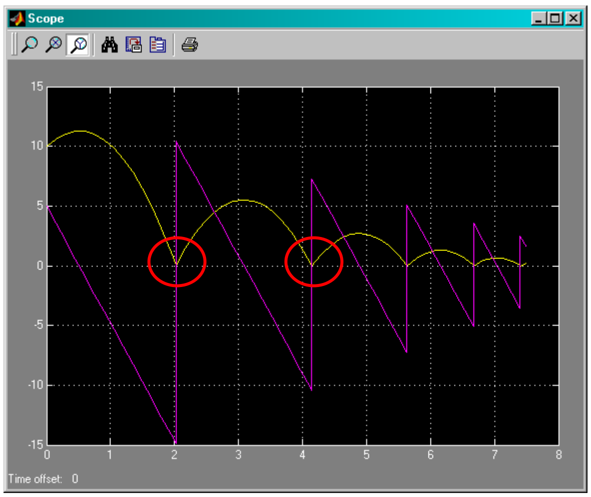
Example #1: Bouncing ball

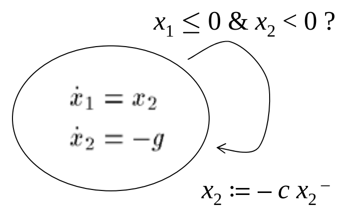


before the transition the linear approximation seems very good so the integration algorithm is fooled into choosing a large integration step

Zero-crossing detection

After a transition is detected, the integration algorithm "goes back in time" to determine where the transition occurred and starts a new integration step at that point.

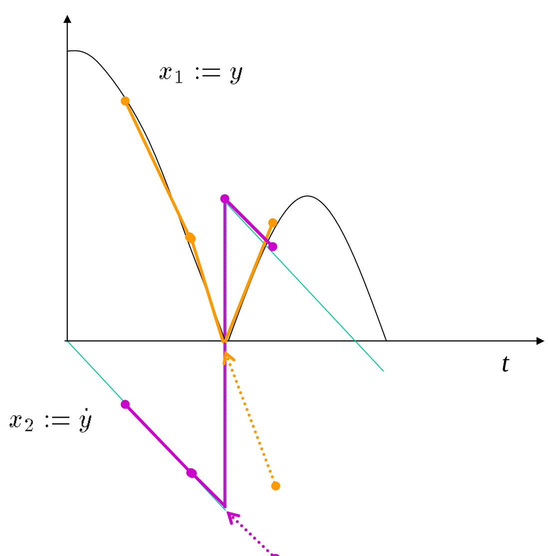




ball_withzerocross.mdl

Zero-crossing detection

After a transition is detected, the integration algorithm "goes back in time" to determine where the transition occurred and starts a new integration step at that point.



Summary

1. Numerical simulation of hybrid automata
   - simulations of ODEs
   - zero-crossing detection

2. Simulators

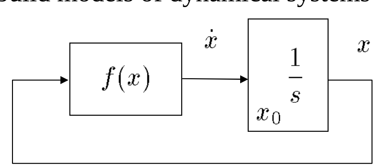
- Simulink { suitable for a small number of discrete modes; difficult to recover hybrid automaton from Simulink file
- Stateflow { good for large numbers of discrete modes and complex transitions; poor integration between continuous and discrete
- SHIFT { very good semantics (easily understandable); poor numerical algorithms
- Modelica { very good numerical algorithms; very convenient for large models with interconnected components; difficult to recover hybrid automaton from simulink file

MATLAB's Simulink

$$\dot{x} = f(x) \qquad x(0) = x_0$$

1. What you see: graphical user interface to build models of dynamical systems

$$x(t) = x_0 + \int_0^t f(x(\tau))d\tau \quad \Leftrightarrow$$



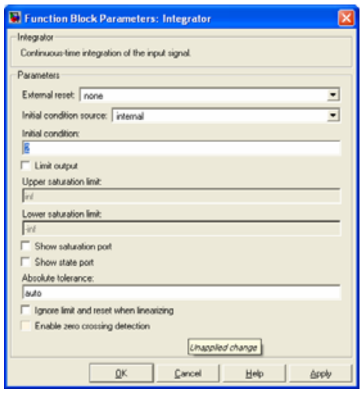
2. What's behind: numerical solver of ODEs with zero-crossing detection

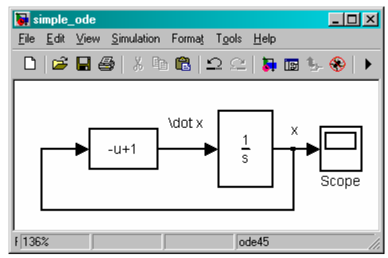
A little history…

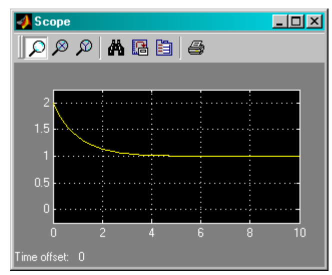
- Commercial product developed by MathWorks
  (founded 1984, flag product is MATLAB/Simulink)
- MATLAB's Simulink was inspired by MATRIXx's SystemBuild
  (in 2001 MathWorks bought MATRIXx)

Simulation of ODEs

$$\dot{x} = -x + 1 \qquad x(0) = 2$$



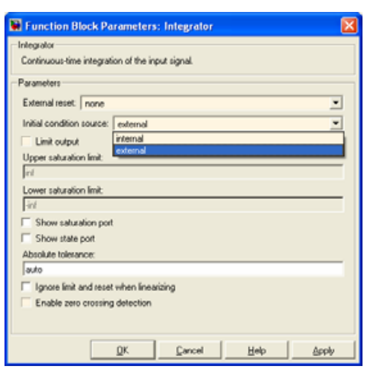


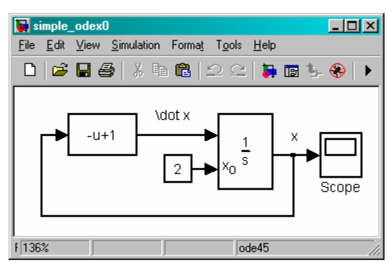


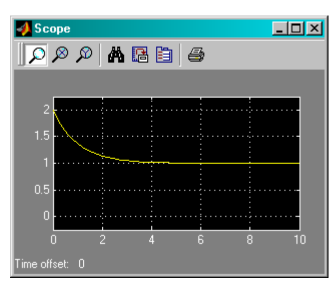
simple_ode.mdl

Simulation of ODEs

$$\dot{x} = -x + 1 \qquad x(0) = 2$$



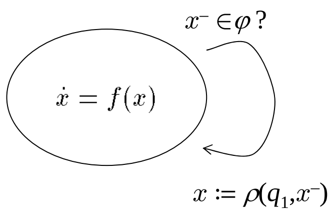




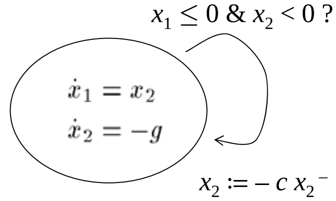
simple_odex0.mdl

ODEs with resets (or impulse systems)

$\mathcal{Q}$ ≡ set of discrete states
$\mathbb{R}^n$ ≡ continuous state-space
$f : \mathbb{R}^n \to \mathbb{R}^n$ ≡ vector field
$\varphi \subset \mathbb{R}^n$ ≡ transition set
$\rho : \mathbb{R}^n \to \mathbb{R}^n$ ≡ reset map

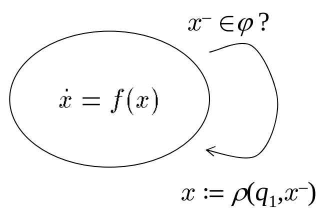


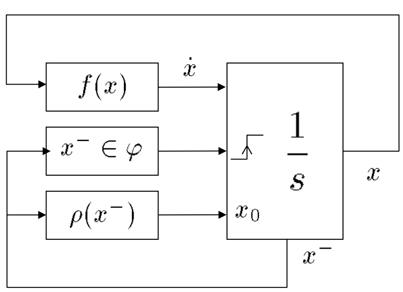
E.g., bouncing ball

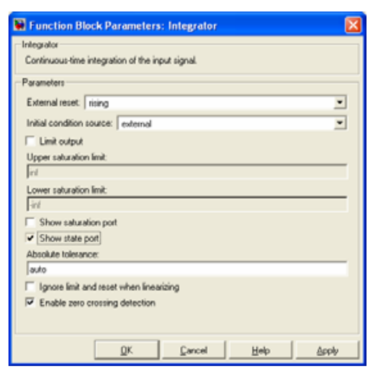


Simulation of ODEs with resets

$\mathcal{Q}$ ≡ set of discrete states
$\mathbb{R}^n$ ≡ continuous state-space
$f : \mathbb{R}^n \to \mathbb{R}^n$ ≡ vector field
$\varphi \subset \mathbb{R}^n$ ≡ transition set
$\rho : \mathbb{R}^n \to \mathbb{R}^n$ ≡ reset map



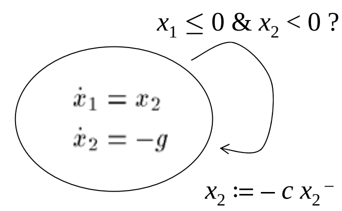


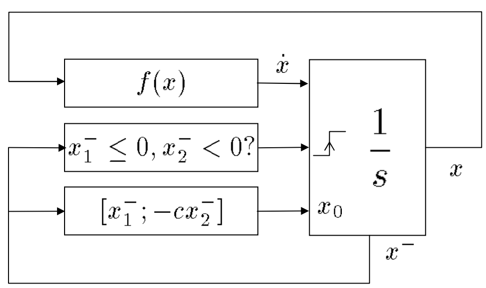


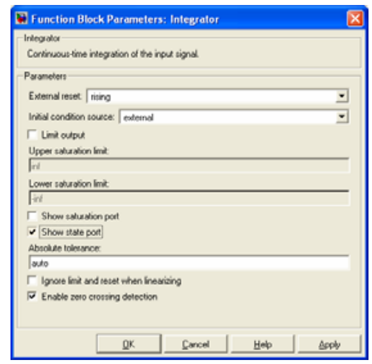
integrator with reset
(by default does zero-crossing
detection on reset input)

Example #1: Bouncing ball

$\mathcal{Q}$ $\equiv$ set of discrete states
$\mathbb{R}^n$ $\equiv$ continuous state-space
$f : \mathbb{R}^n \to \mathbb{R}^n$ $\equiv$ vector field
$\varphi \subset \mathbb{R}^n$ $\equiv$ transition set
$\rho : \mathbb{R}^n \to \mathbb{R}^n$ $\equiv$ reset map

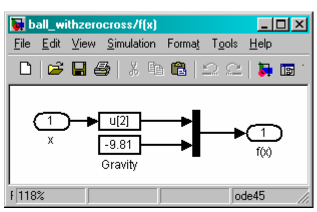


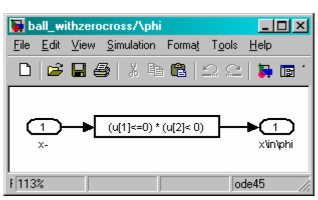


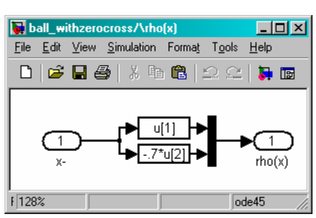


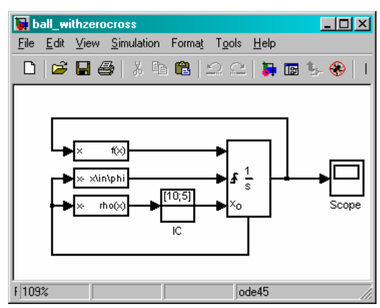
integrator with reset
(by default does zero-crossing
detection on reset input)

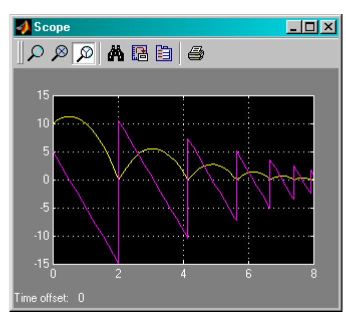
Example #1: Bouncing ball







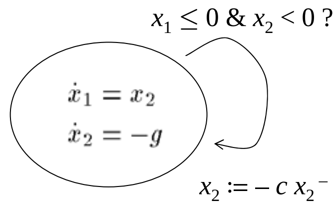


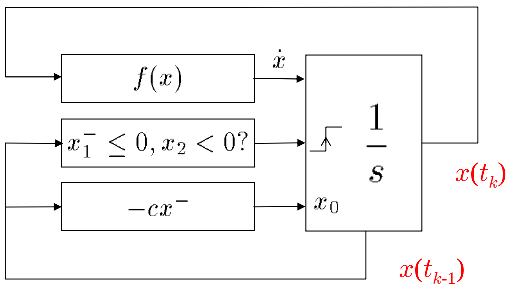


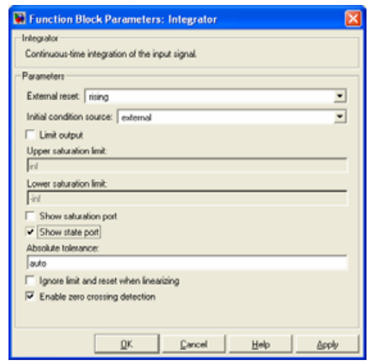
ball_withzerocross.mdl

Simulation of ODEs with resets

$\mathcal{Q}$ $\equiv$ set of discrete states
$\mathbb{R}^n$ $\equiv$ continuous state-space
$f : \mathbb{R}^n \to \mathbb{R}^n$ $\equiv$ vector field
$\varphi \subset \mathbb{R}^n$ $\equiv$ transition set
$\rho : \mathbb{R}^n \to \mathbb{R}^n$ $\equiv$ reset map

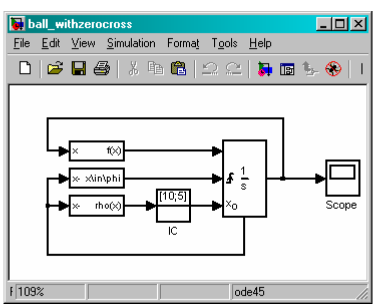




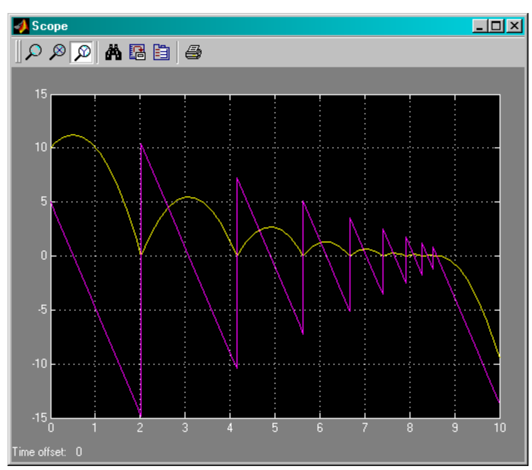


The "reset" pulse is not really instantaneous
(may lead to problems for "Zeno" systems)

Example #1: Bouncing ball



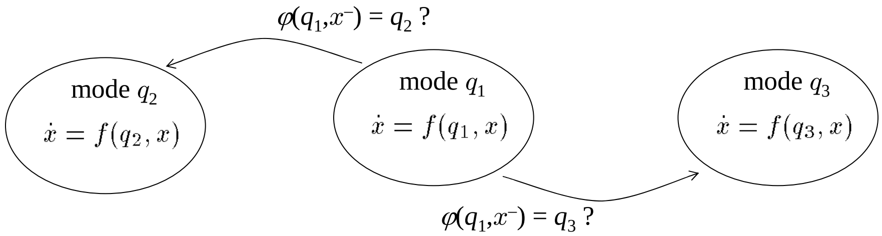
fails to catch the "rising edge" of the reset trigger and the ball falls



ball_withzerocross.mdl

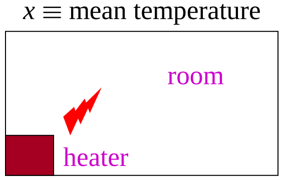
Simulation of hybrid automaton (no resets)

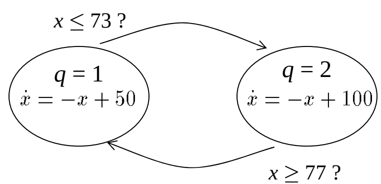
$\mathcal{Q}$ $\equiv$ set of discrete states
$\mathbb{R}^n$ $\equiv$ continuous state-space
$f : \mathcal{Q} \times \mathbb{R}^n \to \mathbb{R}^n$ $\equiv$ vector field
$\varphi : \mathcal{Q} \times \mathbb{R}^n \to \mathcal{Q}$ $\equiv$ discrete transition



E.g., thermostat

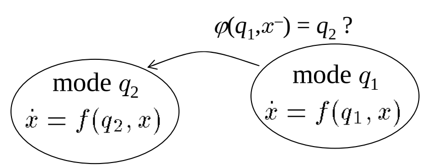
$x \equiv$ mean temperature

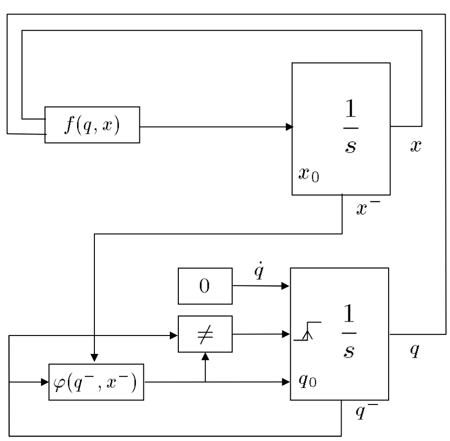




Simulation of hybrid automaton (no resets)

$\mathcal{Q}$ $\equiv$ set of discrete states
$\mathbb{R}^n$ $\equiv$ continuous state-space
$f : \mathcal{Q} \times \mathbb{R}^n \to \mathbb{R}^n$ $\equiv$ vector field
$\varphi : \mathcal{Q} \times \mathbb{R}^n \to \mathcal{Q}$ $\equiv$ discrete transition

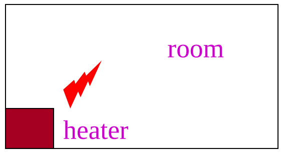


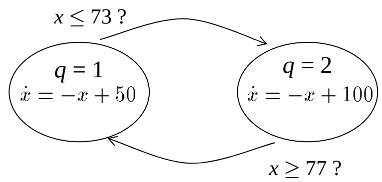


$\mathcal{Q} \subset \mathbb{R}$

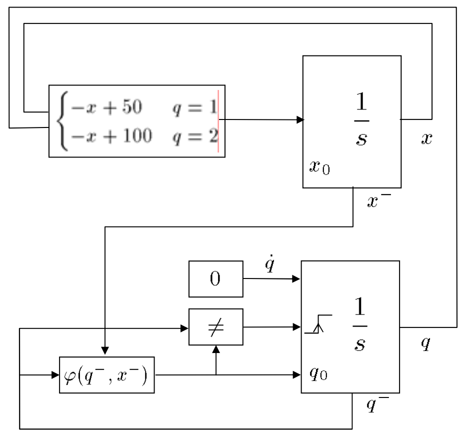
Example #2: Thermostat

$x \equiv$ mean temperature

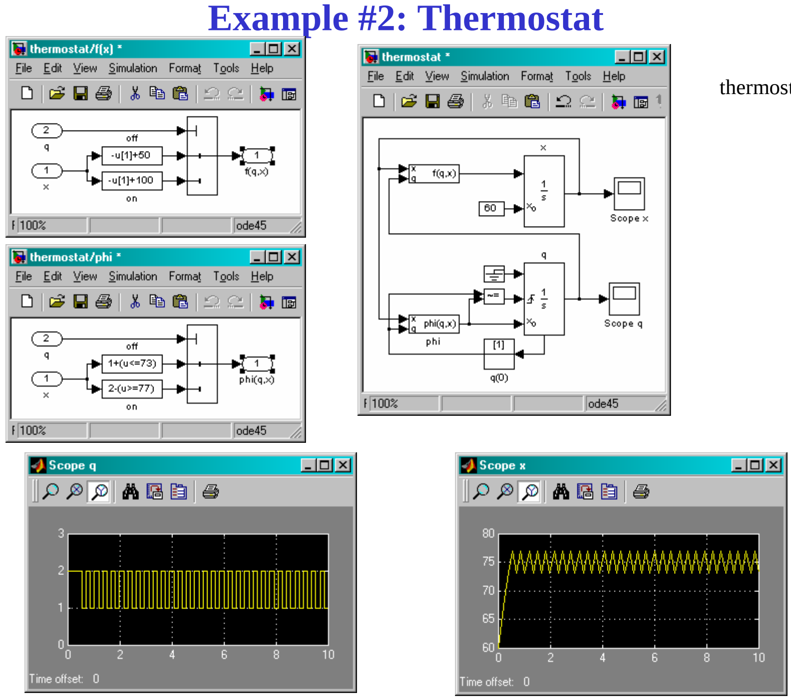




$$\varphi(q, x) := \begin{cases} 2, & q = 1, x \le 73 \\ 1 & q = 1, x > 73 \\ 1, & q = 2, x \ge 77 \\ 2, & q = 2, x < 77 \end{cases}$$



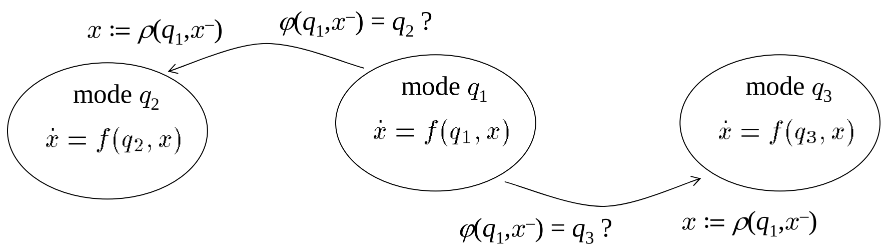
Example #2: Thermostat



thermostat.mdl

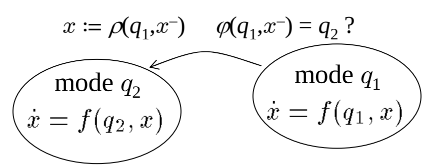
Simulation of hybrid automaton

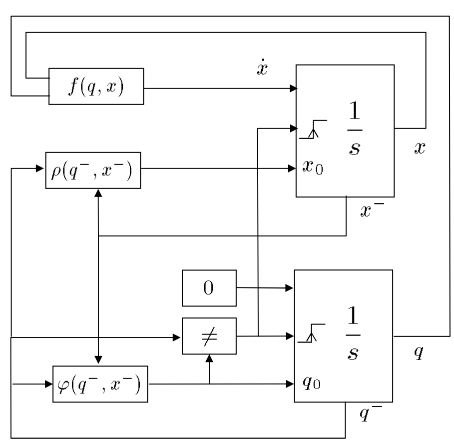
$\mathcal{Q}$ ≡ set of discrete states
$\mathbb{R}^n$ ≡ continuous state-space
$f : \mathcal{Q} \times \mathbb{R}^n \to \mathbb{R}^n$ ≡ vector field
$\varphi : \mathcal{Q} \times \mathbb{R}^n \to \mathcal{Q}$ ≡ discrete transition
$\rho : \mathcal{Q} \times \mathbb{R}^n \to \mathbb{R}^n$ ≡ reset map



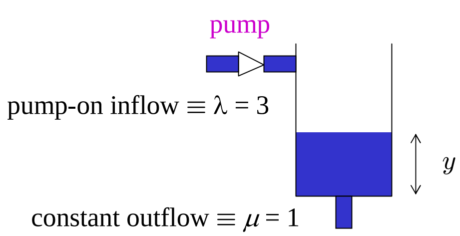
Simulation of hybrid automaton

$\mathcal{Q}$ ≡ set of discrete states
$\mathbb{R}^n$ ≡ continuous state-space
$f : \mathcal{Q} \times \mathbb{R}^n \to \mathbb{R}^n$ ≡ vector field
$\varphi : \mathcal{Q} \times \mathbb{R}^n \to \mathcal{Q}$ ≡ discrete transition
$\rho : \mathcal{Q} \times \mathbb{R}^n \to \mathbb{R}^n$ ≡ reset map



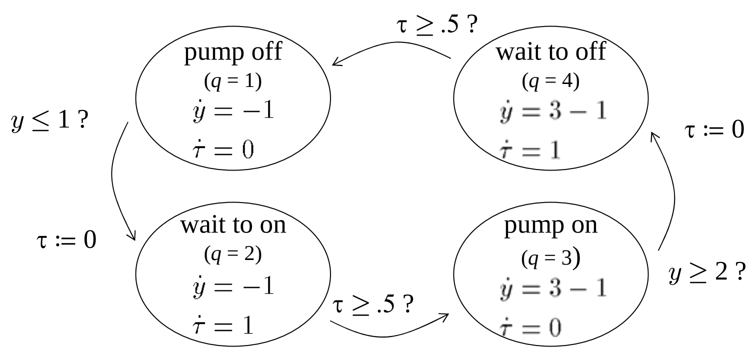


Example #5: Tank system

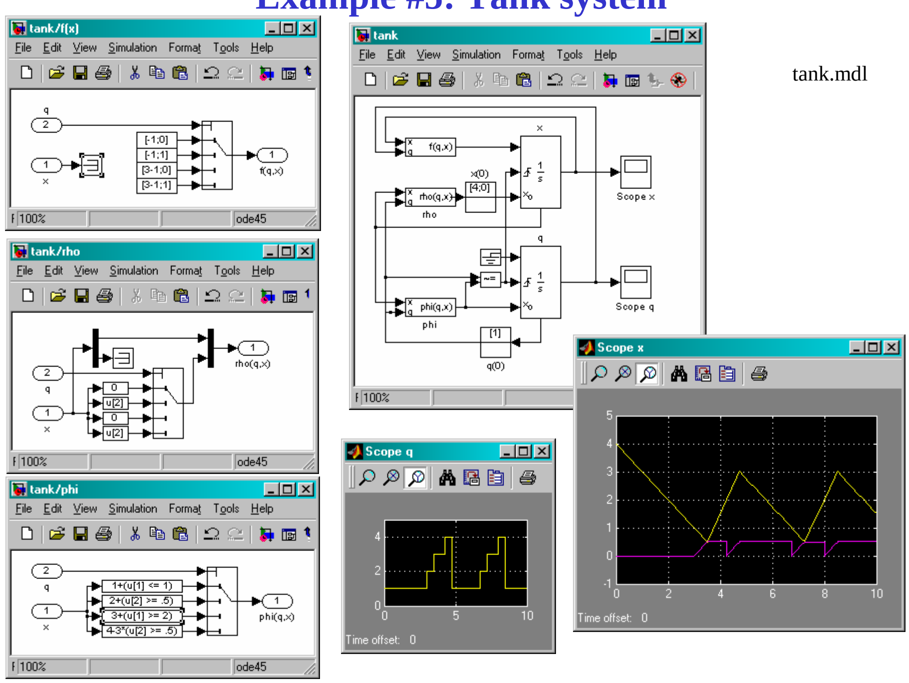


goal ≡ prevent the tank from emptying or filling up

δ = .5 ≡ delay between command is sent to pump and the time it is executed



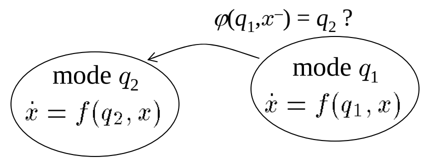
Example #5: Tank system

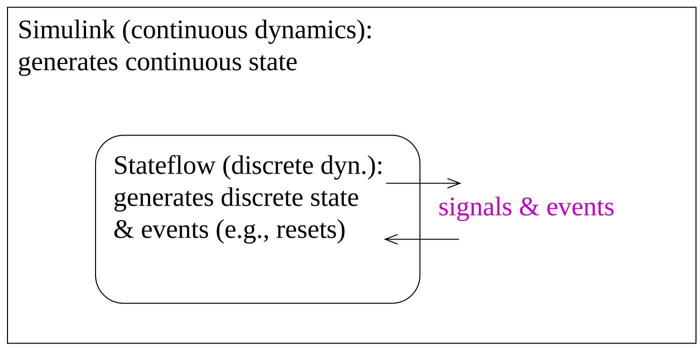


tank.mdl

Stateflow

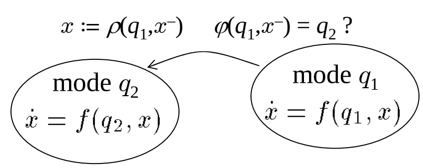
$\mathcal{Q}$ $\equiv$ set of discrete states
$\mathbb{R}^n$ $\equiv$ continuous state-space
$f : \mathcal{Q} \times \mathbb{R}^n \to \mathbb{R}^n$ $\equiv$ vector field
$\varphi : \mathcal{Q} \times \mathbb{R}^n \to \mathcal{Q}$ $\equiv$ discrete transition

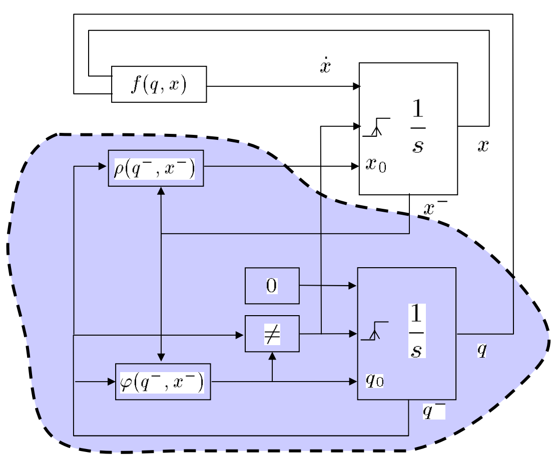




Simulation of hybrid automaton

$\mathcal{Q}$ $\equiv$ set of discrete states
$\mathbb{R}^n$ $\equiv$ continuous state-space
$f : \mathcal{Q} \times \mathbb{R}^n \to \mathbb{R}^n$ $\equiv$ vector field
$\varphi : \mathcal{Q} \times \mathbb{R}^n \to \mathcal{Q}$ $\equiv$ discrete transition
$\rho : \mathcal{Q} \times \mathbb{R}^n \to \mathbb{R}^n$ $\equiv$ reset map

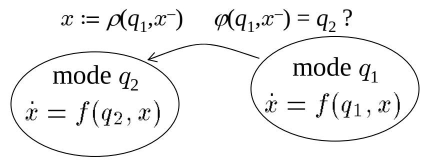


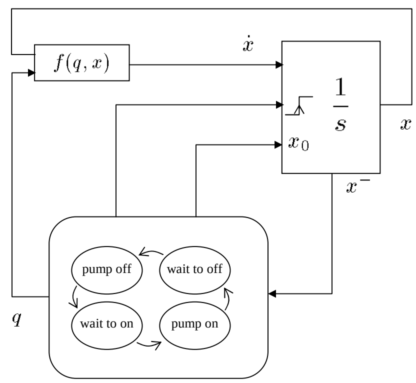


replaced by
StateFlow
"chart"

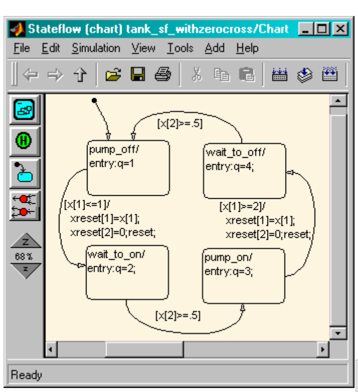
Simulation of hybrid automaton

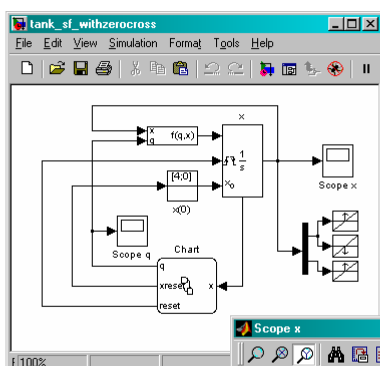
$\mathcal{Q}$ ≡ set of discrete states
$\mathbb{R}^n$ ≡ continuous state-space
$f : \mathcal{Q} \times \mathbb{R}^n \to \mathbb{R}^n$ ≡ vector field
$\varphi : \mathcal{Q} \times \mathbb{R}^n \to \mathcal{Q}$ ≡ discrete transition
$\rho : \mathcal{Q} \times \mathbb{R}^n \to \mathbb{R}^n$ ≡ reset map



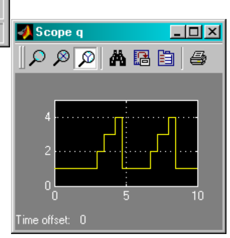


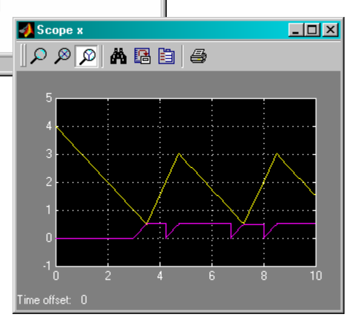
Example #5: Tank system





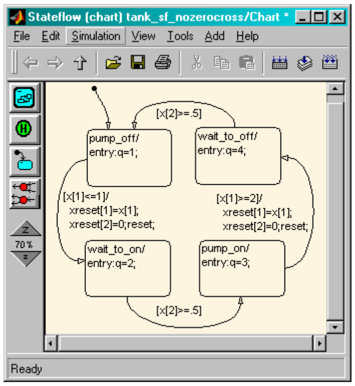
Stateflow does not automatically does zero crossing detection, which must be done "manually"

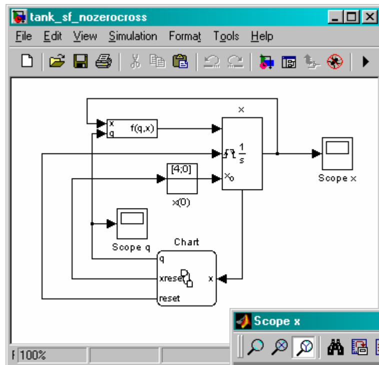




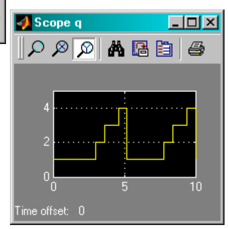
tank_sf_withzerocross.mdl

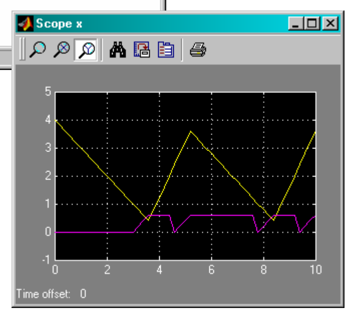
Example #5: Tank system





Stateflow does not automatically does zero crossing detection, which must be done "manually"





tank_sf_nozerocross.mdl

SHIFT

(Hybrid System Tool Interchange Format)

1. Simulation Language for Hybrid Automaton
2. Developed at UC Berkeley under the PATH (Partners for Advance Transit and Highways) project to simulate Automated Highway Systems
3. PATH provides freeware to compile SHIFT into a C-based simulator (http://path.berkeley.edu/SHIFT/)

Advantages:

1. Syntax matches very closely the hybrid automata formalism
2. Object-oriented and scalable (multiple copies of a automata can be created, connected, disconnected, and destroyed as the simulation runs)

Problems:

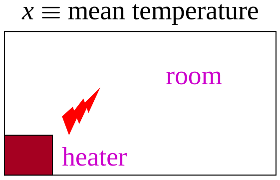
1. Currently the simulation engine is poor (limited integration algorithms, no zero-crossing detection)
2. Still in the development stages

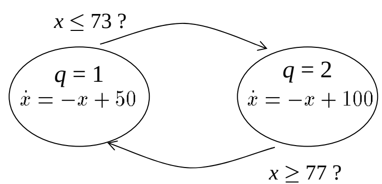
A SHIFT program

```
type AutomataType {   // declaration of automata types

      input …          // declaration of input signals
      state …          // declaration of internal states
      output …         // declaration of output signals
      export …         // declaration of events for synchronization

      discrete …       // declaration of discrete modes
                       // & corresponding continuous dynamics
      transition …     // definition of transition rules

      setup …          // initializations
}

global …               // declaration of global Automata

setup …                // creation and initialization of global Automata
```

Example #2: Thermostat

$x \equiv$ mean temperature





thermostat.hs

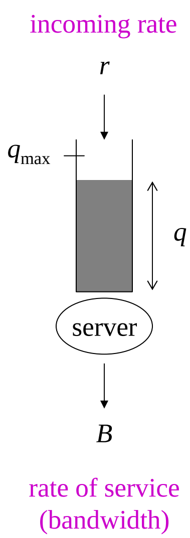
```
type ThermostatType {
     state continuous number x := 50;
     discrete                          first declared is the initial mode
          heater_off {  x' = -x + 50; },
          heater_on  {  x' = -x + 100; };
     transition
          heater_off -> heater_on  {} when { x<= 73},
          heater_on  -> heater_off {} when { x>= 77};
}

global ThermostatType thermostat;

setup
     define {  thermostat := create(ThermostatType, x:= 60);  }
```

declaration of the ThermostatType type

declaration of a variable called "thermostat" of type "ThermostatType"

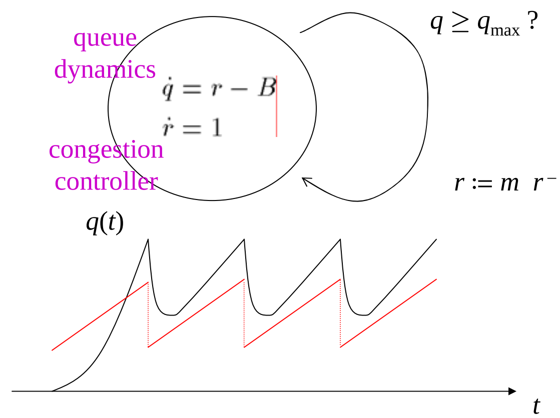
creation (and initialization) of the hybrid automaton

Example #7: Server system with congestion control

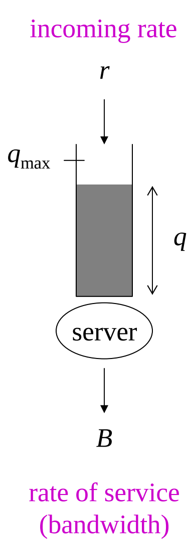


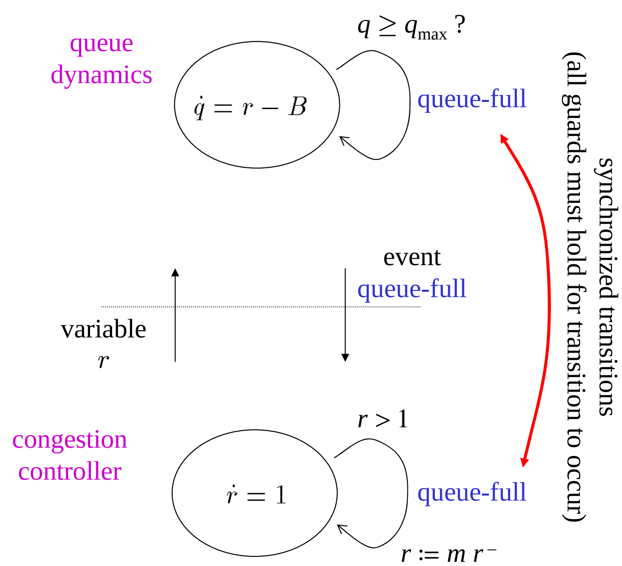
Additive increase/multiplicative decrease congestion control (AIMD):

- while $q < q_{max}$ increase r linearly
- when q reaches q_{max} instantaneously multiply r by $m \in (0,1)$

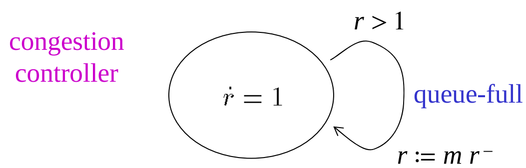


Example #7: Server system with congestion control





Example #7: Server system with congestion control

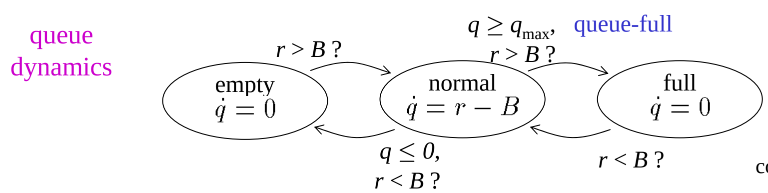


congestion.hs

```
type CongestionControllerType {
	output continuous number r := 0;

	state number m := 0.5;			// multiplicative decrease (parameter)

	export queue_full;

	discrete
		additive_increase   { r' = 1; }

	transition
		additive_increase -> additive_increase { queue_full } when { r > 1 } do { r := m * r; }
}
```

synchronization event | jump condition | reset

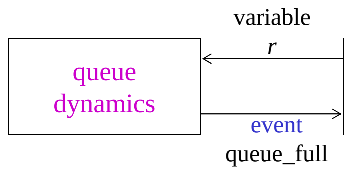
Example #7: Server system with congestion control



congestion.hs

```
type QueueType {
	input continuous number r := 0;
	state continuous number q := 0;
	state CongestionControllerType controller;		// controller to synchronize with

	state number B := 1;			// Bandwidth (parameter)
	state number qmax := 10;		// Maximum queue size (parameter)

	discrete
		empty_queue { q' = 0; }
		full_queue  { q' = 0; }
		normal { q' = r - B; }

	transition
		empty_queue -> normal {} when { r > B },
		full_queue -> normal  {} when { r < B };
		normal -> empty_queue {} when { q <= 0 and r <= B ) do { q = 0; },
		normal -> full_queue  { controller:queue_full } when { q >= Qmax and r>B ) do { q = qmax; },
}
```

Example #7: Server system with congestion control



congestion.hs

```
global QueueType  queue;
global CongestionControllerType contr;

setup
        define {
                contr := create(CongestionControllerType);
                queue := create(QueueType, controller := contr);
        }
        connect {
                r(queue) <- r(contr);
        }
```

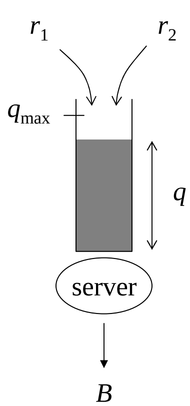
declaration of global variables

creation (and initialization) of the hybrid automata

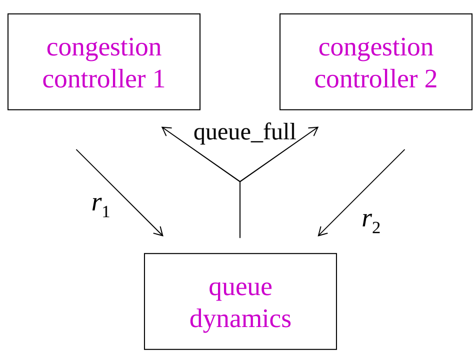
inputs ← output connections

Example #10: Server with multiple congestion controllers

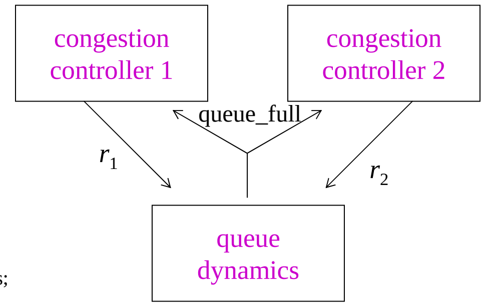
incoming rates



rate of service
(bandwidth)



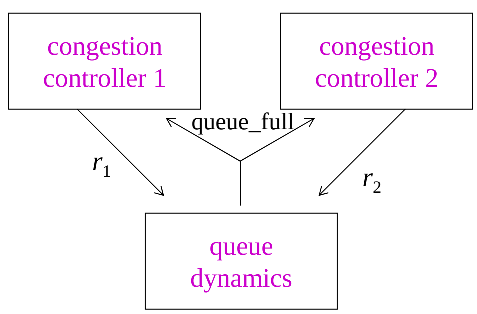
Example #10: Server with multiple congestion controllers




```
type QueueType {
        input continuous number r1 := 0;
        input continuous number r2 := 0;
        state continuous number r := 0;
        state continuous number q := 0;
        state set(CongestionControllerType) controllers;

        state number B := 1;           // Bandwidth (parameter)
        state number qmax := 10;       // Maximum queue size (parameter)

        discrete
                empty_queue { q’ = 0; r = r1 + r2; }
                full_queue  { q’ = 0; r = r1 + r2; }
                normal { q’ = r - B; r = r1 + r2; }

        transition
                empty_queue -> normal {} when { r > B },
                full_queue -> normal  {} when { r < B };
                normal -> empty_queue {} when { q <= 0 and r <= B ) do { q = 0; },
                normal -> full_queue  { controllers:queue_full(all) } when { q >= Qmax and r>B ) do { q = qmax; },
}
```

congestion2.hs

Example #10: Server with multiple congestion controllers




```
global QueueType  queue;
global CongestionControllerType contr1;
global CongestionControllerType contr2;
```

declaration of global variables

congestion2.hs

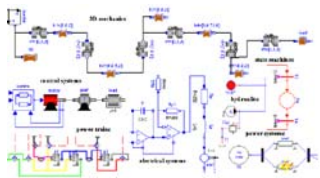
```
setup
        define {
                contr1 := create(CongestionControllerType, r := 1.5);
                contr2 := create(CongestionControllerType, r := 2.0);
                queue := create(QueueType, controllers := {contr1,contr2} );
        }
        connect {
                r1(queue) <- r(contr1);
                r2(queue) <- r(contr2);
        }
```

creation (and initialization) of the hybrid automata

inputs ← output connections

Modelica

1. Object-oriented language for modeling physical systems
2. Developed and promoted by the Modelica Association
   (international non-profit, non-governmental organization based in Sweden)
   http://www.modelica.org/
3. Dynasim AB sells Dymola, currently the best Modelica simulator
   (interfaces with MATLAB and Simulink)



Advantages:
1. Object-oriented and scalable
2. Large number of component libraries available (electric circuits, mechanical system, thermo-hydraulics, power systems, robotics, petri-nets, etc.)
3. Numerically stable simulation engine
   (allows differential algebraic equations, automatic zero-crossing detection)
4. Heavily used in industry, especially in Europe
5. UCSB has a site license!

Problems:
1. Not as widely known as MATLAB/Simulink

Modelica object

```
model ModelName
    // declarations of public variables (inputs and outputs) that
    // can be accessed using “ModelName.Variable name”
    TypeName VariableName;
    …
protected
    // declarations of internal (hidden) variables (state)
    TypeName VariableName;
    …
equation
    // algebraic and differential equations
    Expression = Expression;
    …
end ModelName
```

Modelica 101

Predefined types

```
Real       x,       // x is a real number
           y (start = 1.1, unit = "inches" ) "height";
                    // y is a "height," initialized with 1.1 "inches"
Integer    n;       // n is an integer
Boolean    p;       // p can be either true or false
```

Type modifiers

```
parameter  Real B;  // B does not change during the simulation but
                    // can be initialized with different values
constant   Real PI; // PI is a fixed constant
discrete   Real q;  // q is a piecewise constant variable (discrete state)
flow       Real r;  // r is a "flow" variable
                    // (connection of flows follow conservation law,
                    // by convention positive means flow enters component)
```

Modelica 101

Predefined functions/variables for use in equation

der(x) // derivative of *Real* signal x

pre(x) // left-limit of *discrete* signal x

edge(x) // true when *discrete* variable x is discontinuous

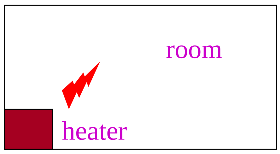
time // simulation time

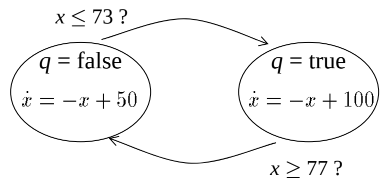
Commands for use in equation

```
x = y              // equate two variable
x + z = y          // (should be interpreted as equation and not assignment)
reinit(x, 2.1);    // reset variable x to to 2.1
connect(x,y);      // connects variables:
connect(x,z);      // x = y = z (or x + y + z = 0 in case of flows)
when p then        // execute command when p becomes true
    command;
end when;
```

Example #2: Thermostat

$x \equiv$ mean temperature






```
model thermostat

    Real x                      "Average temperature";
    Boolean q(start = false)    "Heater state";

    parameter Real xon = 73  "Turn-on temperature";
    parameter Real xoff = 77 "Turn-off temperature";

equation
    q =  if  not pre(q) and x <= xon then true   // turn on
         else if pre(q) and x >= xoff then false // turn off
         else pre(q);                            // no change
    der(x) = if q then 100-x
                  else 50-x;
end thermostat;
```

thermostat.mo

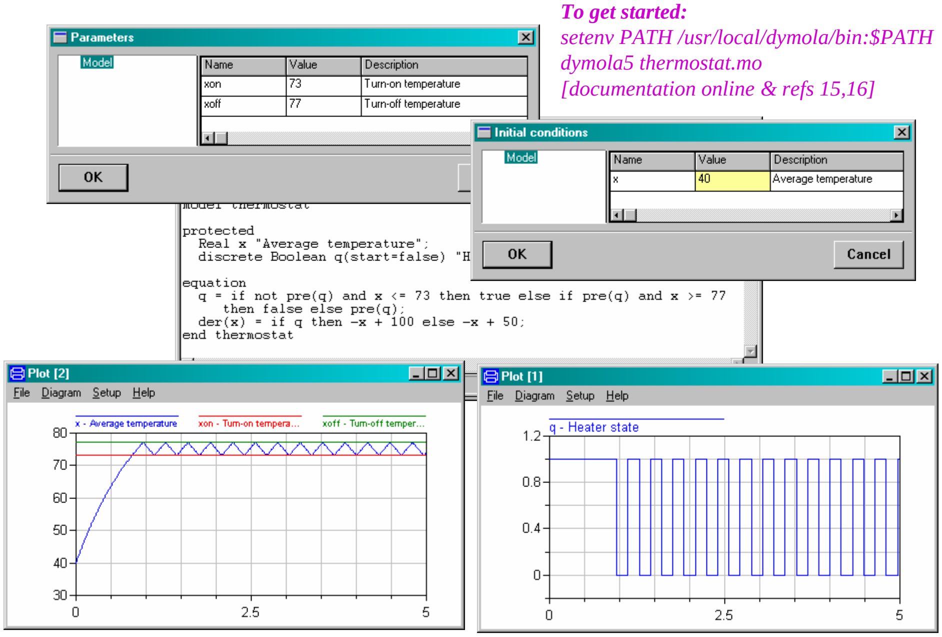
Example #2: Thermostat

To get started:

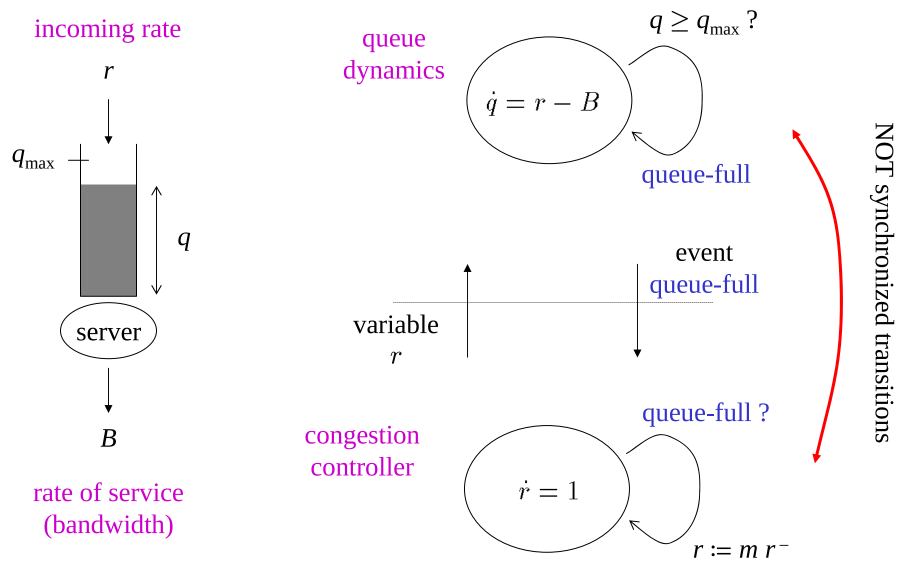
setenv PATH /usr/local/dymola/bin:$PATH

dymola5 thermostat.mo

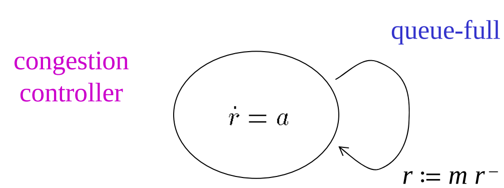
[documentation online & refs 15,16]



Example #7: Server system with congestion control



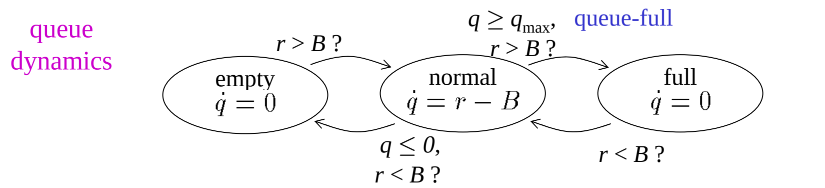
Example #7: Server system with congestion control




```
model Controller
	Real r(start=0)			"out-flow";
	discrete Boolean queue_full	"Queue full";
	parameter Real a=.1		"Additive constant";
	parameter Real m=.5		"Multiplicative constant";

equation
	der(r) = a;
	when pre(queue_full) then
		reinit(r, r*m);
	end when;

end Controller
```

congestion.mo

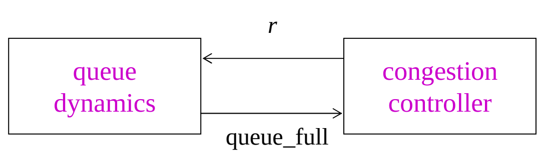
Example #7: Server system with congestion control




```
model Queue
	Real r "In-flow";
	discrete Boolean queue_full(start=false) "Queue full";
	parameter Real B=1                       "Bandwidth";
	parameter Real Qmax=1                    "Max queue size";

protected
	discrete Integer status(start=0)         "Queue status";    // 0 empty, 1 normal, 2 full
	Real q(start=0)                          "Queue size";
equation
	status = if pre(status) == 0 and r > B then 1                        // no longer empty
		else if pre(status) == 2 and r < B then 1                         // no longer full
		else if pre(status) == 1 and q <= 0 and r < B then 0              // became empty
		else if pre(status) == 1 and q >= Qmax and r > B then 2           // became full
		else pre(status);                                                 // no change
	queue_full = pre(status) == 1 and status == 2;
	der(q) = if status == 1 then r - B else 0;
end Queue
```

congestion.mo

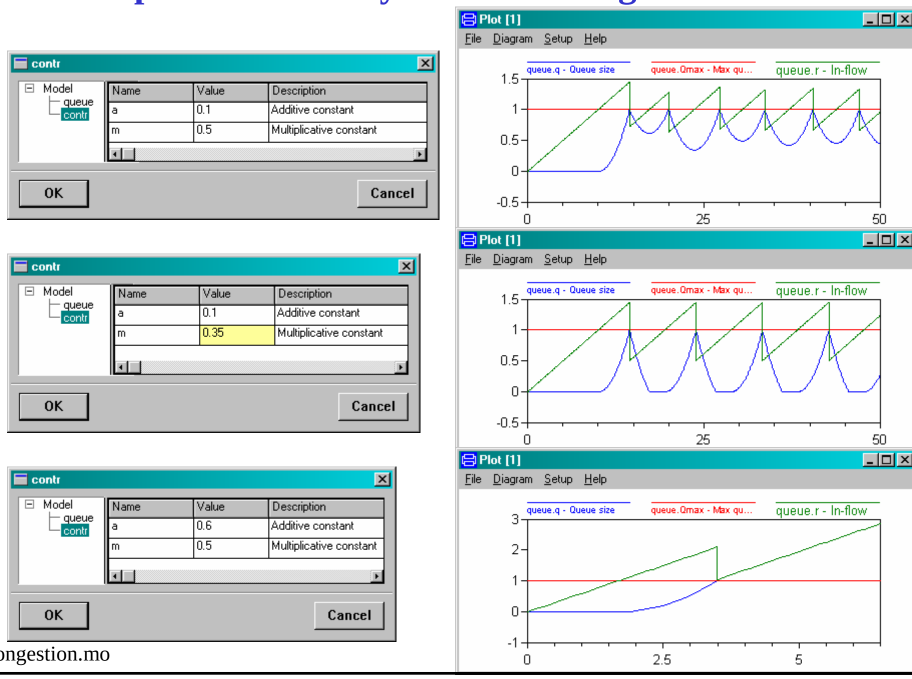
Example #7: Server system with congestion control




```
model System
	Queue queue;
	Controller contr;

equation
	queue.r = contr.r;
	queue.queue_full = contr.queue_full;

end System
```

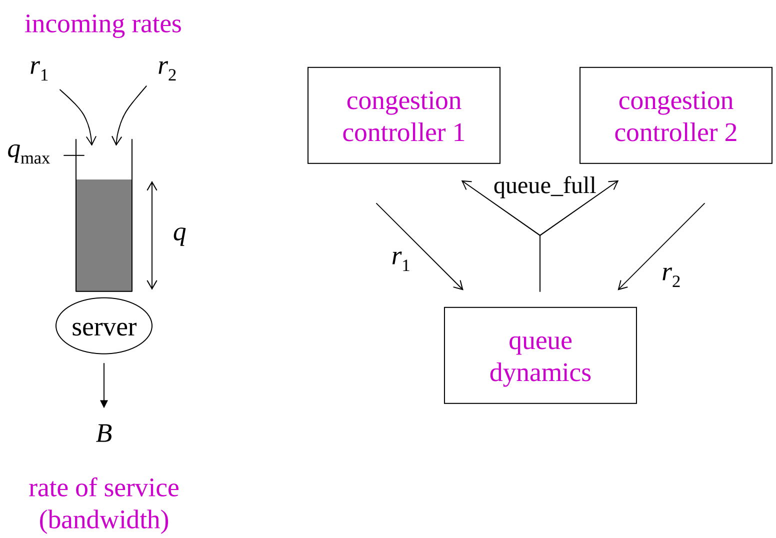
congestion.mo

Example #7: Server system with congestion control



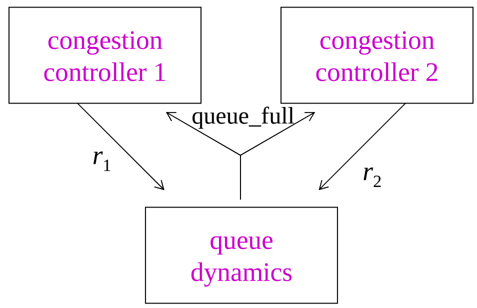
congestion.mo

Example #10: Server with multiple congestion controllers



congestion2.mo

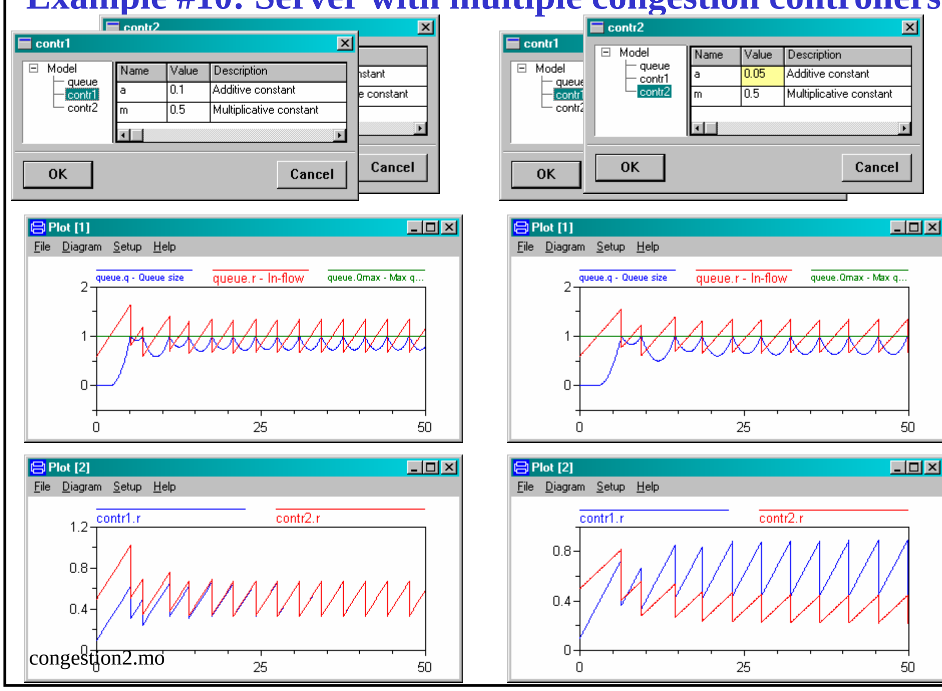
Example #10: Server with multiple congestion controllers




```
model System
      Queue queue;
      Controller contr1;
      Controller contr2;

equation
      queue.r = contr1.r + contr2.r;
      queue.queue_full = contr1.queue_full;
      queue.queue_full = contr2.queue_full;

end System
```

congestion2.mo

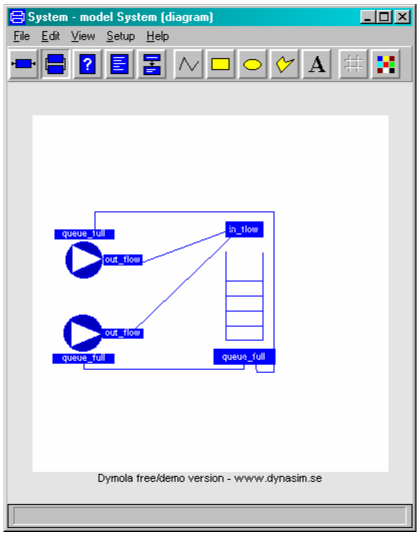
Example #10: Server with multiple congestion controllers

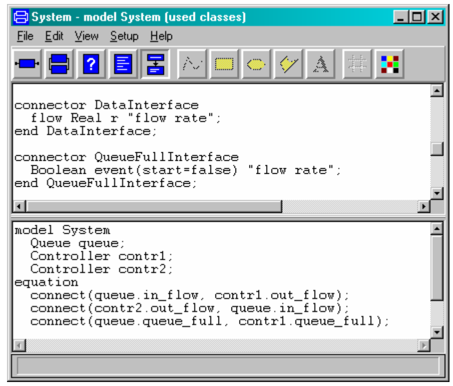


congestion2.mo

Example #10: Server with multiple congestion controllers

see congestion3.mo for the use of the connect command





congestion3.mo

Interfacing Dymola & MATLAB/Simulink

Sharing data

- Each time you simulate a Dymola model, a file “xxxx.mat” is created with all the data. You can get this data into MATLAB with
  load xxxx.mat

Using Dymola models in Simulink

- Add dymola to the MATLAB path:
  path('/usr/local/dymola/mfiles',path)
  path('/usr/local/dymola/mfiles/traj',path)
- The Simulink block
  /usr/local/dymola/mfiles/DymolaBlockMaster.mdl
  can be used to insert a Dymola system into a Simulink diagram
  Not sure if its working?

Next class…

Properties of hybrid systems

- Safety (reachability)
- Liveness
- Asymptotic properties (stability)