Concurrent Models of Computation for Embedded Software

Edward A. Lee

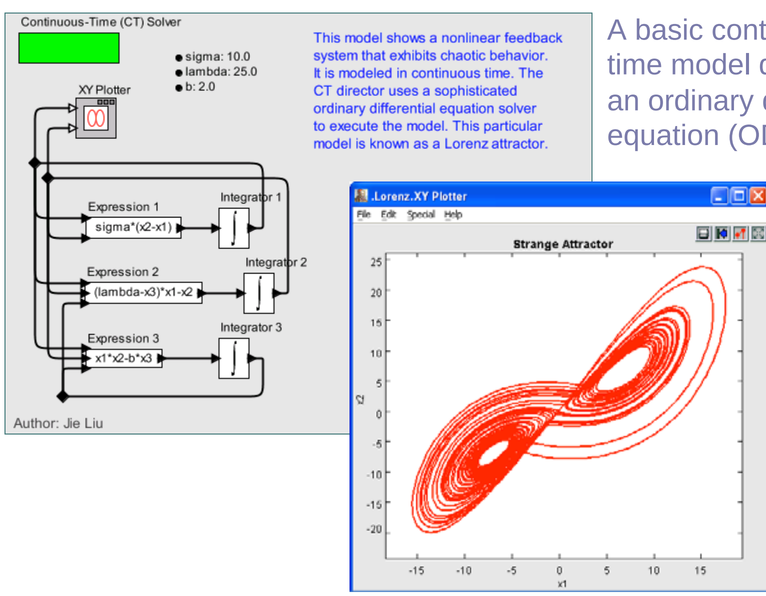
Professor, UC Berkeley

EECS 290n – Advanced Topics in Systems Theory

Spring, 2009

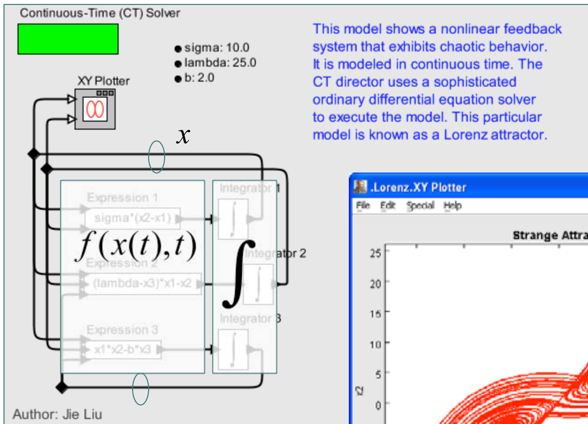
Lecture 14: Continuous-Time and Hybrid Systems

Basic Continuous-Time Modeling

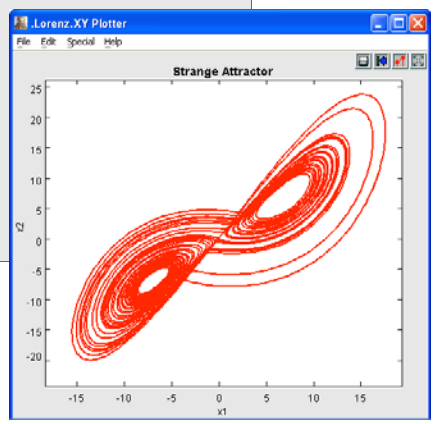


A basic continuous-time model describes an ordinary differential equation (ODE).

Basic Continuous-Time Modeling



A basic continuous-time model describes an ordinary differential equation (ODE).



$$\dot{x}(t) = f(x(t),t)$$

$$x(t) = x(t_0) + \int_{t_0}^{t} \dot{x}(\tau)d\tau$$

Basic Continuous-Time Modeling

The state trajectory is modeled as a vector function of time,

$$x : T \to R^n \qquad T = [t_0, \infty) \subset R$$

$$f(x(t),t) \xrightarrow{\dot{x}} x(t) = x(t_0) + \int_{t_0}^{t} \dot{x}(\tau)d\tau \xrightarrow{x}$$

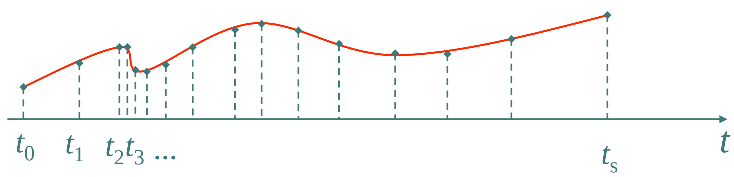
$$\dot{x}(t) = f(x(t),t)$$

$$f : R^m \times T \to R^m$$

ODE Solvers

Numerical solution approximates the state trajectory of the ODE by estimating its value at discrete time points:

$$\{t_0, t_1, \ldots\} \subset T$$



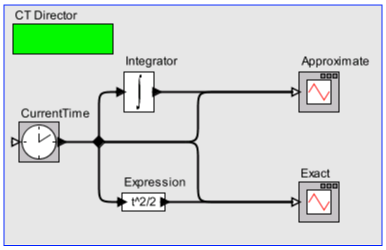
Reasonable choices for these points depend on the function *f*.

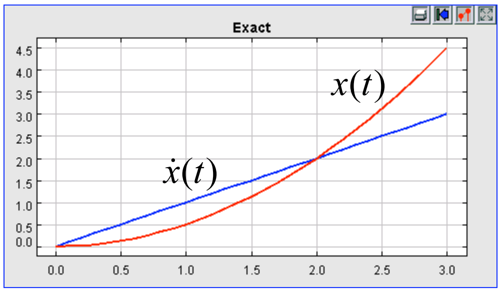
Using such solvers, signals are discrete-event signals.

Simple Example

This simple example integrates a ramp, generated by the CurrentTime actor. In this case, it is easy to find a closed form solution,

$$\dot{x}(t) = t \;\Rightarrow\; x(t) = t^2/2$$

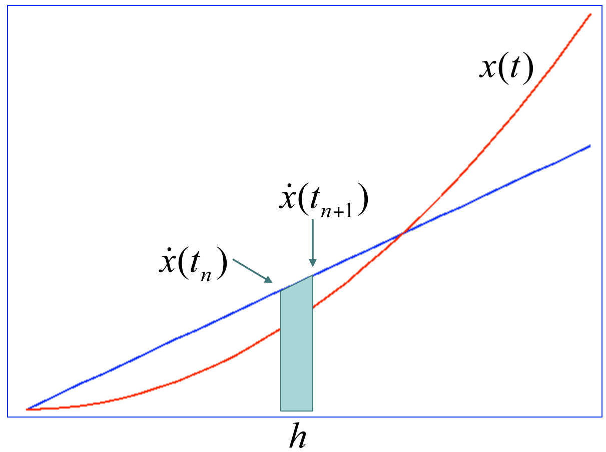




Trapezoidal Method

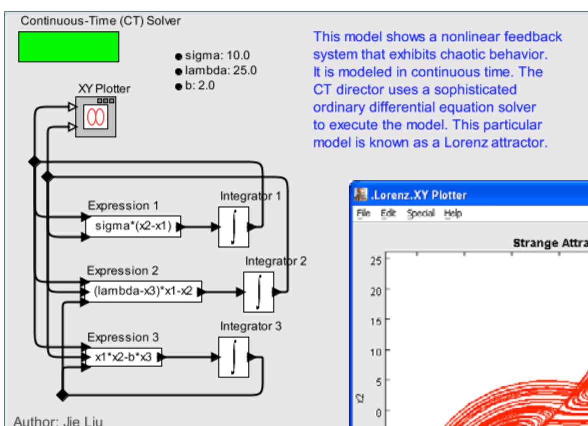
Classical method estimates the area under the curve by calculating the area of trapezoids.

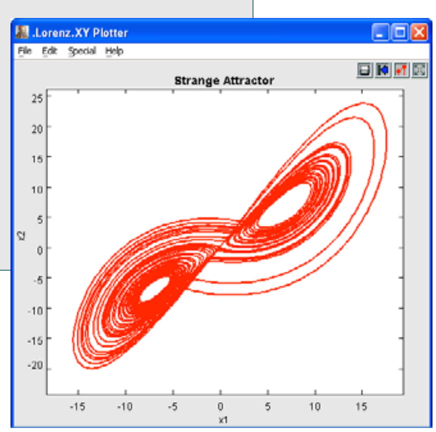
However, with this method, an integrator is only causal, not strictly causal or delta causal.



$$x(t_{n+1}) = x(t_n) + h(\dot{x}(t_n) + \dot{x}(t_{n+1}))/2$$

Trapezoidal Method is Problematic with Feedback





We have no assurance of a unique fixed point, nor a method for constructing it.

Forward Euler Solver

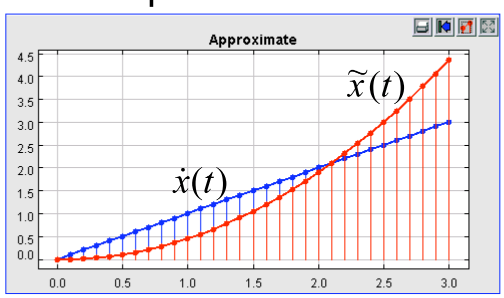
Given $x(t_n)$ and a time increment h, calculate:

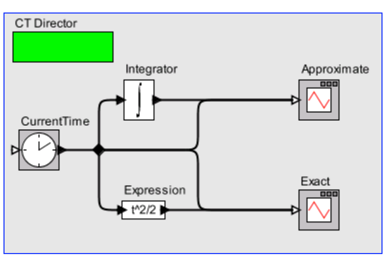
$$t_{n+1} = t_n + h$$
$$x(t_{n+1}) = x(t_n) + h\, f(x(t_n), t_n)$$

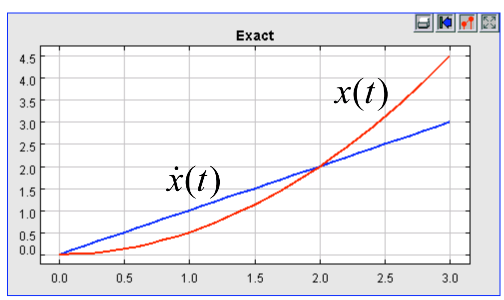
This method is strictly causal, or, with a lower bound on the step size h, delta causal. It can be used in feedback systems. The solution is unique and non-Zeno.

Forward Euler on Simple Example

In this case, we have used a fixed step size $h = 0.1$. The result is close, but diverges over time.







"Stiff" systems require small step sizes

Force due to spring extension:

$$F_1(t) = k(p - x(t))$$

Force due to viscous damping:

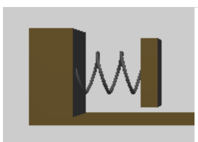
$$F_2(t) = -c\dot{x}(t)$$

Newton's second law:

$$F_1(t) + F_2(t) = M\ddot{x}(t)$$

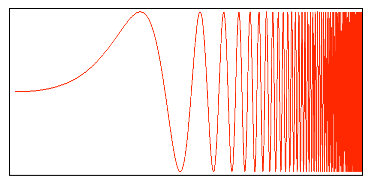
or

$$M\ddot{x}(t) + c\dot{x}(t) + kx(t) = kp.$$



For spring-mass damper, large stiffness constant *k* makes the system "stiff."

Variable step-size methods will dynamically modify the step size *h* in response to estimates of the integration error. Even these, however, run into trouble when stiffness varies over time. Extreme case of increasing stiffness results in Zeno behavior:



Runge-Kutta 2-3 Solver (RK2-3)

Given $x(t_n)$ and a time increment h, calculate

$$K_0 = f(x(t_n), t_n) \quad \dot{x}(t_n)$$

$$K_1 = f(x(t_n) + 0.5hK_0, t_n + 0.5h) \quad \text{estimate of } \dot{x}(t_n + 0.5h)$$

$$K_2 = f(x(t_n) + 0.75hK_1, t_n + 0.75h) \quad \text{estimate of } \dot{x}(t_n + 0.75h)$$

then let

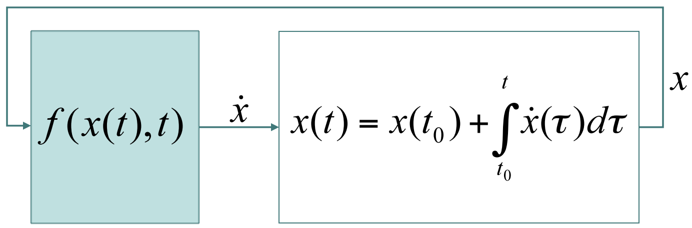
$$t_{n+1} = t_n + h$$

$$x(t_{n+1}) = x(t_n) + (2/9)hK_0 + (3/9)hK_1 + (4/9)hK_2$$

Note that this is strictly (delta) causal, but requires three evaluations of f at three different times with three different inputs.

Operational Requirements

In a software system, the blue box below can be specified by a program that, given $x(t)$ and t calculates $f(x(t), t)$. But this requires that the program be functional (have no side effects).



$$\dot{x}(t) = f(x(t),t)$$

$$f : R^m \times T \rightarrow R^m$$

For variable-step size RK2-3, have to be able to evaluate f at t_n , $t_n + 0.5h$, and $t_n + 0.75h$ without committing to the step size h . (Evaluation must have no side effects).

Adjusting the Time Steps

For time step given by $t_{n+1} = t_n + h$, let

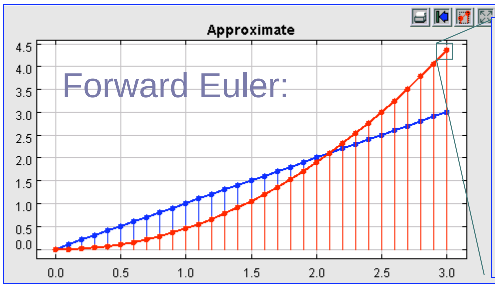
$$K_3 = f(x(t_{n+1}), t_{n+1})$$
$$\varepsilon = h((-5/72)K_0 + (1/12)K_1 + (1/9)K_2 + (-1/8)K_3)$$

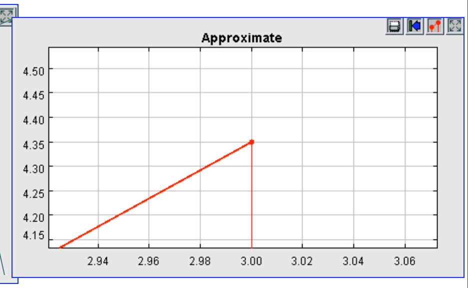
If ε is less than the "error tolerance" e, then the step is deemed "successful" and the next time step is estimated at:

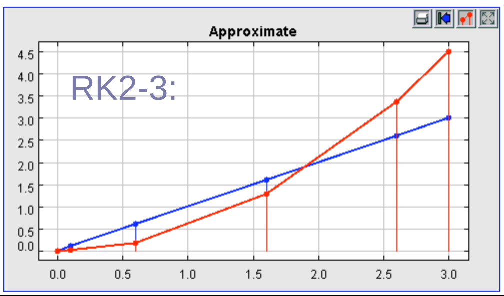
$$h' = 0.8 \sqrt[3]{e/\varepsilon}$$

If ε is greater than the "error tolerance," then the time step h is reduced and the whole thing is tried again.

Comparing RK2-3 to Forward Euler







For this example, RK2-3 is exact at 3.0, while Forward Euler undershoots by a significant amount.

Accumulating Errors

In feedback systems, the errors of FE accumulate more rapidly than those of RK2-3.

$$f(x(t),t) \xrightarrow{\dot{x}} x(t) = x(t_0) + \int_{t_0}^{t} \dot{x}(\tau)d\tau \xrightarrow{x}$$

$$\dot{x}(t) = f(x(t),t)$$

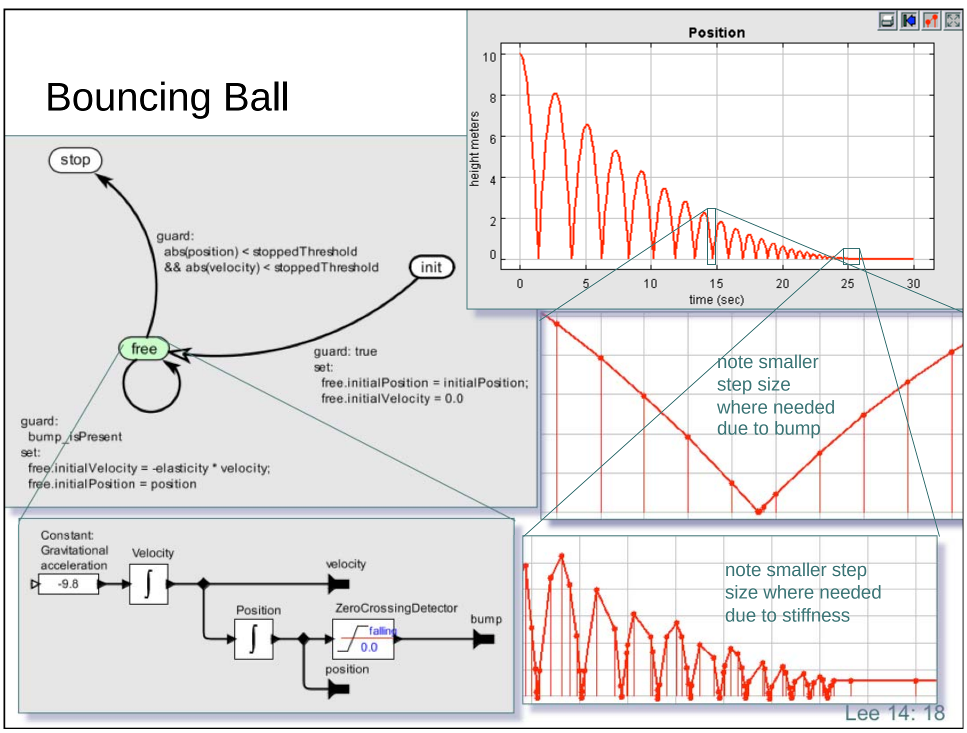
$$f : R^m \times T \to R^m$$

Adjusting the Time Steps due to Discrete Events

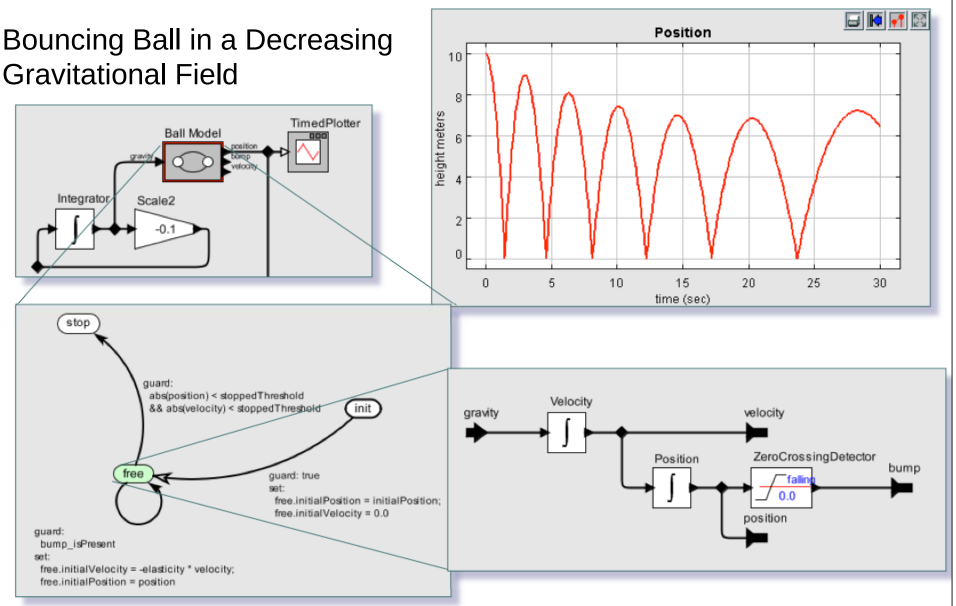
A step size *h* may cause the model to skip over a point where the behavior of the system changes abruptly:

Such events must be detected and treated similarly as requiring a smaller step size.

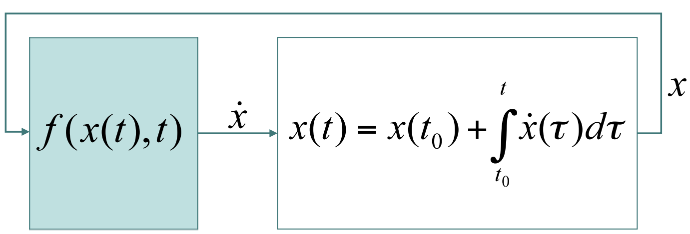
Bouncing Ball



Bouncing Ball in a Decreasing Gravitational Field



Examining This Computationally

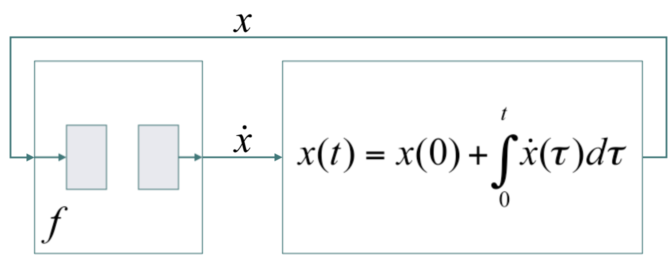


At each discrete time t_n, given a time increment $t_{n+1} = t_n + h$, we can estimate $x(t_{n+1})$ by repeatedly evaluating f with different values for the arguments. We may then decide that h is too large and reduce it and redo the process.

How General Is This MoC?

Does it handle:

- Systems without feedback? yes
- External inputs? yes
- State machines?

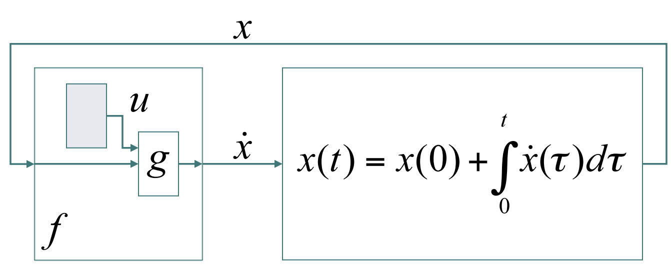


$$\dot{x}(t) = f(x(t), t)$$

How General Is This MoC?

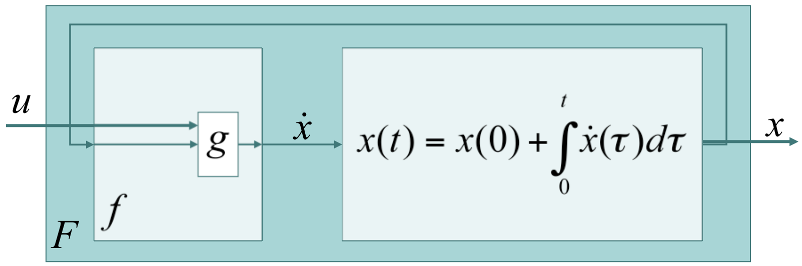
Does it handle:

- Systems without feedback?
- External inputs? yes
- State machines?



$$\dot{x}(t) = f(x(t), t) = g(u(t), x(t), t)$$

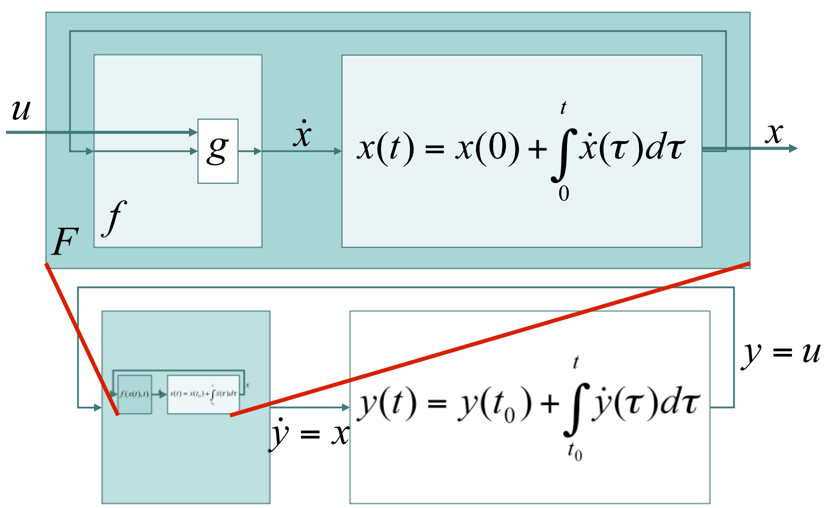
The Model Itself as a Function



Note that the model function has the form:

$$F : (T \to R^m) \to (T \to R^m)$$

Is the MoC Compositional?



For a model of computation to be *compositional*, it must be possible to turn a model into a component in another model.

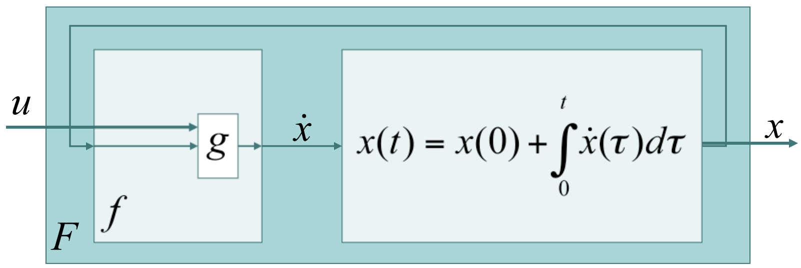
The Model Itself as a Function

Note that the model function has the form:

$$F : (T \to R^m) \to (T \to R^m)$$

Which does not match the form:

$$f : R^m \times T \to R^m$$



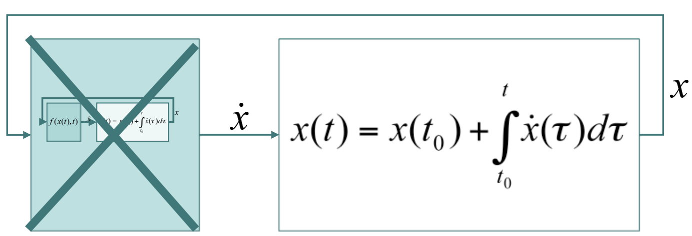
Given the model, we don't actually know the function *f*.

Lee 14: 25

Consequently, the MoC is Not Compositional!

In general, the behavior of the inside dynamical system cannot be given by a function of form:

$$f : R^m \times T \to R^m$$



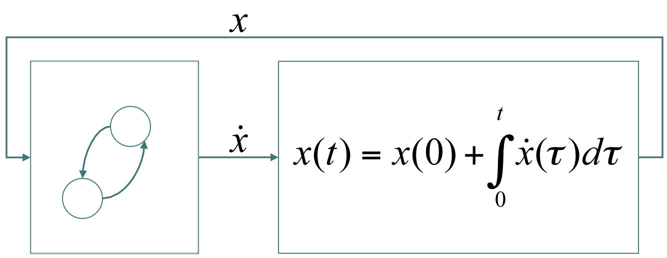
To see this, just note that the output must depend only on the current value of the input and the time to conform with this form.

Lee 14: 26

So How General Is This MoC?

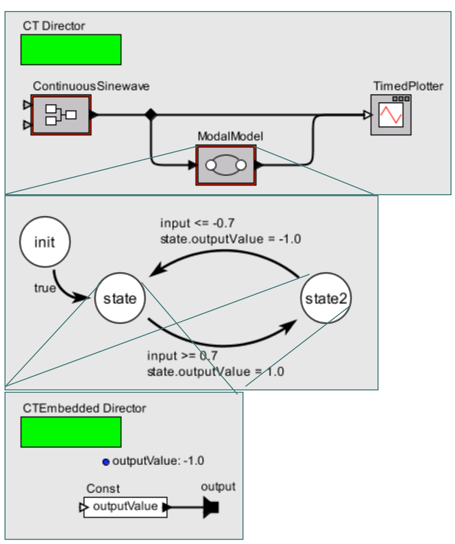
Does it handle:

- External inputs?
- Systems without feedback?
- State machines? No… The model needs work…



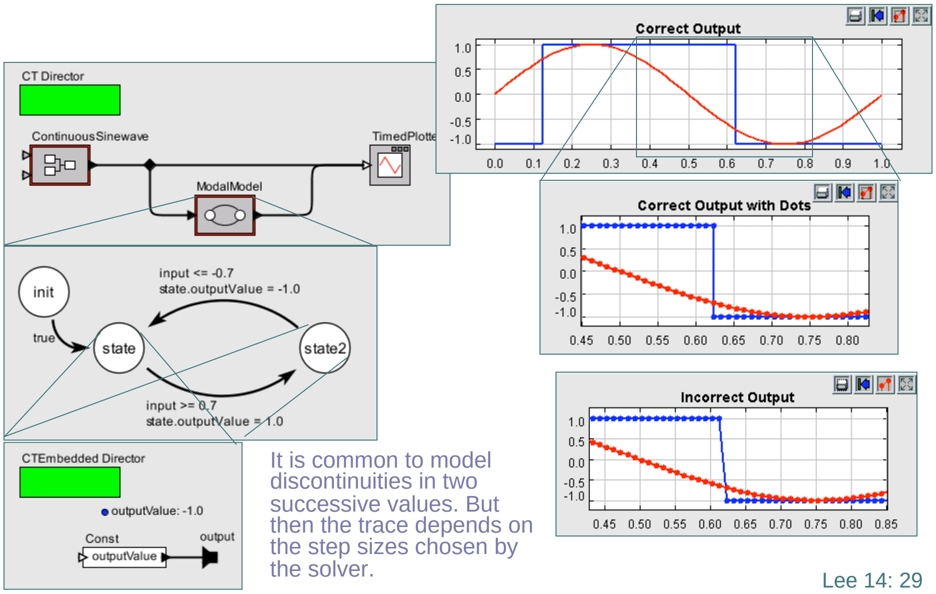
Since this model is itself a state machine, the inability to put a state machine in the left box explains the lack of compositionality.

Start with Simple State Machines
Hysteresis Example



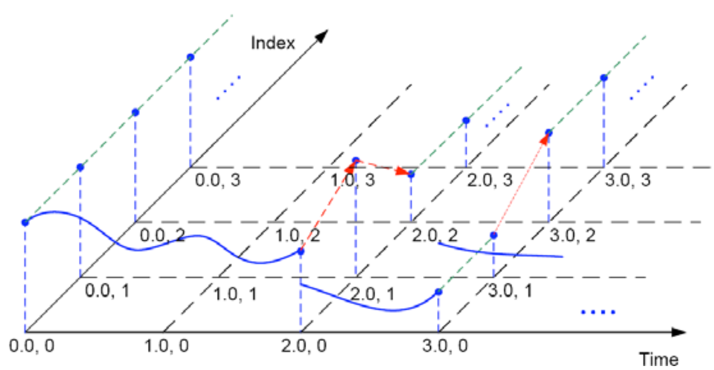
This model shows the use of a two-state FSM to model hysteresis. Semantically, the output of the ModalModel block is discontinuous. If transitions take zero time, this is modeled as a signal that has two values *at the same time*, and *in a particular order*.

Hysteresis Example Requires Superdense Time



It is common to model discontinuities in two successive values. But then the trace depends on the step sizes chosen by the solver.

Recall Superdense Time



At each tag, the signal has **exactly one value**. At each time point, the signal has **an infinite number of values**. The red arrows indicate value changes between tags, which correspond to discontinuities.

Initial and Final Value Signals

A signal $x: T \times \mathbb{N} \to V$ has no *chattering Zeno* condition if there is an integer $m > 0$ such that

$$\forall n > m, \quad x(t,n) = x(t,m)$$

A non-chattering signal has a corresponding *final value signal*, $x_f: T \to V$ where

$$\forall\, t \in T, \quad x_f(t) = x(t,m)$$

It also has an *initial value signal* $x_i: T \to V$ where

$$\forall\, t \in T, \quad x_i(t) = x(t,0)$$

Piecewise Continuous Signals

A piecewise continuous signal is a non-chattering signal

$$x: T \times \mathbb{N} \to V$$

where

- The initial signal x_i is continuous on the left,
- The final signal x_f is continuous on the right, and
- The signal x has only one value at all $t \in T \setminus D$ where $D \subset T$ is a discrete set.

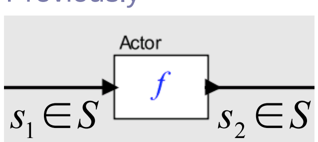
Requirements

The hysteresis example illustrates two requirements:

- A signal may have more than one value at a particular time, and the values it has have an order.
- The times at which the solver evaluates signals must precisely include the times at which interesting events happen, like a guard becoming true.

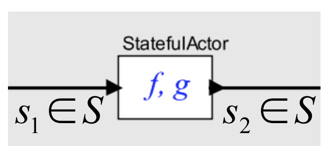
Both Requirements Are Dealt With By an Abstract Semantics

Previously



$$S = [T \to R]$$
$$f : R^m \times T \to R^m$$
$$\forall t \in T, \quad s_2(t) = f(s_1(t), t)$$

Now we need:



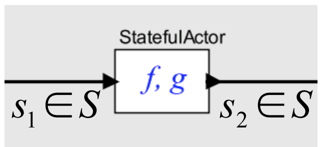
$$S = [T \times N \to R]$$
$$f : \Sigma \times R^m \times T \to R^m$$
$$g : \Sigma \times R^m \times T \to \Sigma$$

state space

$$\forall (t,n) \in T \times N, \quad s_2(t,n) = ?$$

The new function *f* gives outputs in terms of inputs and the current state. The function *g* updates the state at the specified time.

Abstract Semantics



$$S = [T \times N \to R]$$
$$f : \Sigma \times R^m \times T \to R^m$$
$$g : \Sigma \times R^m \times T \to \Sigma$$

At each $t \in T$ the output is a *sequence* of one or more values where given the current state $\sigma(t) \in \Sigma$ and the input $s_1(t)$ we evaluate the procedure

$$s_1(t,0) = f(\sigma(t), s_1(t,0), t)$$
$$\sigma_1(t) = g(\sigma(t), s_1(t,0), t)$$
$$s_2(t,1) = f(\sigma_1(t), s_2(t,1), t)$$
$$\sigma_2(t) = g(\sigma_1(t), s_2(t,1), t)$$
$$\ldots$$

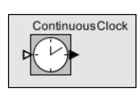
Fixed-point problem

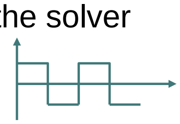
until the state no longer changes. We use the final state on any evaluation at later times.

This deals with the first requirement.

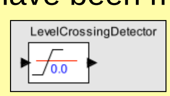
Second Requirement:
Points on the Time Line that Must Be Included in a Discrete Trace

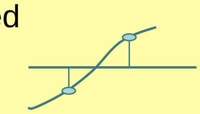
- Predictable breakpoints
  - Can be registered in advance with the solver





- Unpredictable breakpoints
  - Known after they have been missed



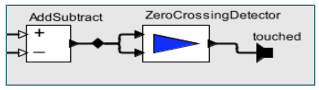


- Points that make the step size “sufficiently small”
  - Dependent on error estimation in the solver

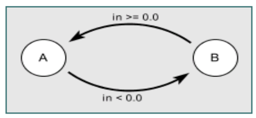
Require backtracking

Event Times

In continuous-time models, Ptolemy II can use *event detectors* to identify the precise time at which an event occurs:

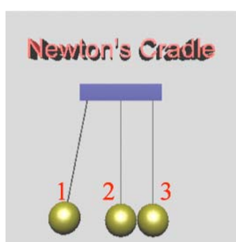


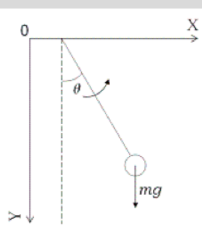
or it can use Modal Models, where guards on the transitions specify when events occur. In the literature, you can find two semantic interpretations to guards: *enabling* or *triggering*.



If only enabling semantics are provided, then it becomes nearly impossible to give models whose behavior does not depend on the step-size choices of the solver.

Another Example: Newton's Cradle



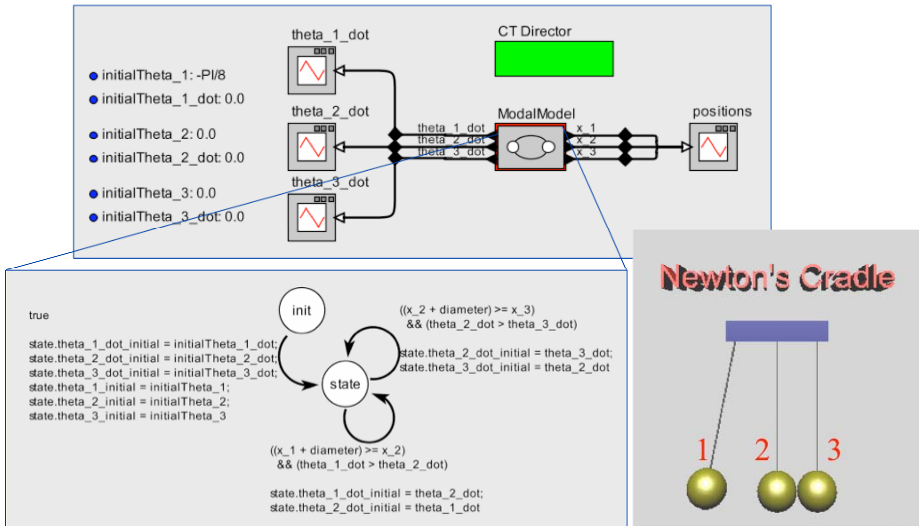


$$ml\ddot{\theta} = -mg\sin(\theta)$$

Assumptions

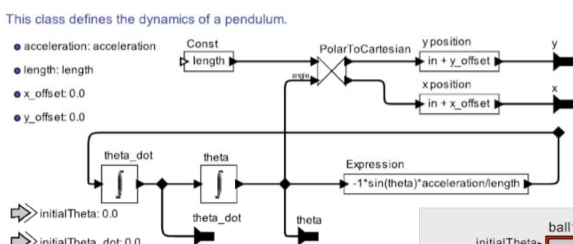
- Ideal pendulum
- Balls have the same mass.
- Collisions happen instantaneously.
- When a collision happens, two and only two balls are involved.

A Model of Newton's Cradle

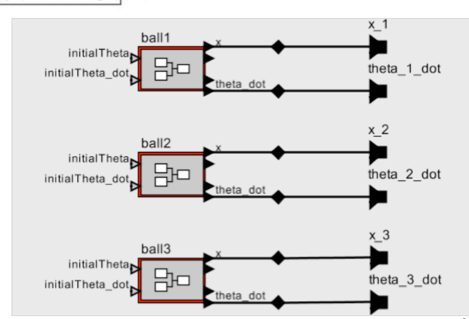


Slide from Haiyang Zheng

Dynamics of Balls



Three second order ODE's are used to model the dynamics of three pendulums.

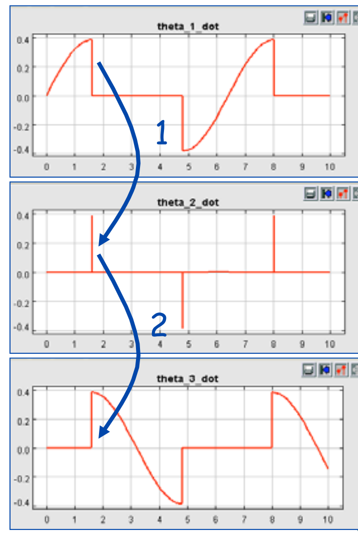


40

Slide from Haiyang Zheng

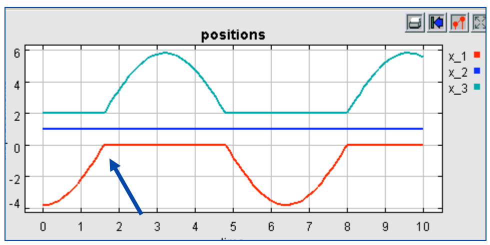
One Behavior

X-axis is time and Y-axis is velocity.



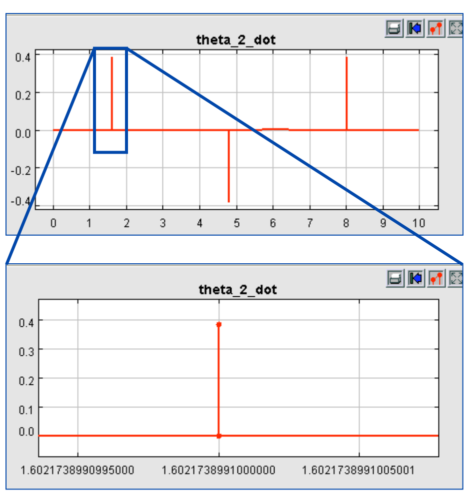
Ball #1 is moved away from its equilibrium position with angle PI/8.

Perfectly elastic collisions.



X-axis is time and Y-axis is displacement.

Interactions Between CT and DE Dynamics

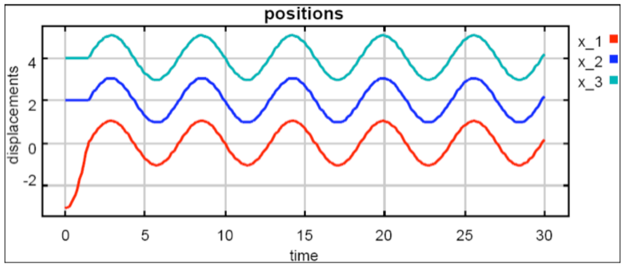


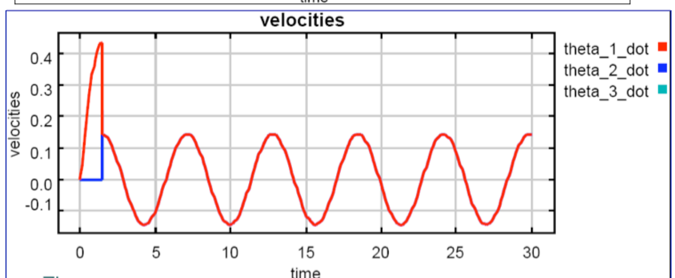
Two transitions at the same time, called simultaneous discrete events.

These events cause a discontinuity consisting of three values.

Agreement on the assumption of instantaneous collisions

Another Behavior:
Perfectly Inelastic Collisions

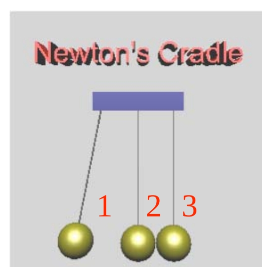


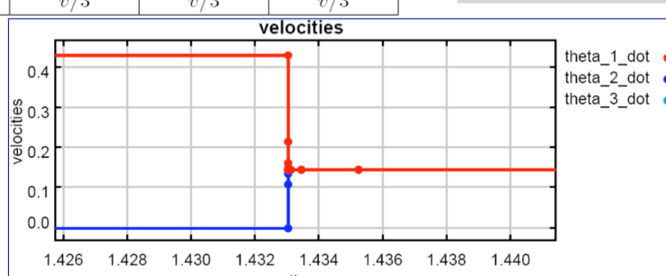


A Zeno Phenomenon

Table 1.1. An infinite sequence of collisions leads to a steady state.

$\#of collisions$	v_1	v_2	v_3
0	v	0	0
1	$v/2$	$v/2$	0
2	$v/2$	$v/4$	$v/4$
3	$3v/8$	$3v/8$	$v/4$
4	$3v/8$	$5v/16$	$5v/16$
5	$11v/32$	$11v/32$	$5v/16$
⋮	⋮	⋮	⋮
∞	$v/3$	$v/3$	$v/3$





Recall Requirements

We have two requirements:

- A signal may have more than one value at a particular time, and the values it has have an order.
- The times at which the solver evaluates signals must precisely include the times at which interesting events happen, like a guard becoming true, or any point of discontinuity in a signal (a time where it has more than one value).

Ideal Solver Semantics

[Liu and Lee, HSCC 2003]

Given an interval $I = [t_i, t_{i+1}]$ and an initial value $x(t_i)$ and a function $f : R^m \times T \to R^m$ that is Lipschitz in x on the interval (meaning that there exists an $L \geq 0$ such that

$$\forall t \in I, \quad \| f(x(t), t) - f(x'(t), t) \| \leq L \| x(t) - x'(t) \|$$

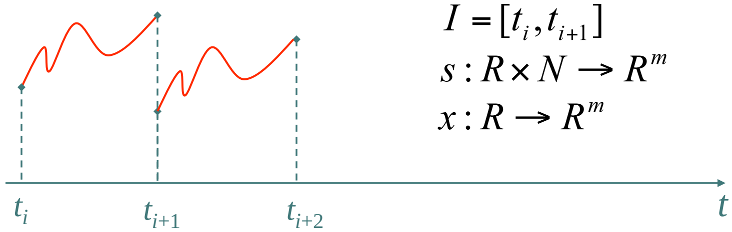
then the following equation has a unique solution x satisfying the initial condition where

$$\forall t \in I, \quad \dot{x}(t) = f(x(t), t)$$

The ideal solver yields the exact value of $x(t_{i+1})$.

Piecewise Lipschitz Systems

In our CT semantics, signals have multiple values at the times of discontinuities. Between discontinuities, a necessary condition that we can impose is that the function f be Lipschitz, where we choose the points at the discontinuities to ensure this:



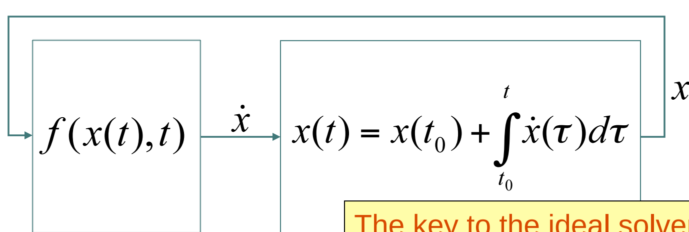
$$I = [t_i, t_{i+1}]$$
$$s : R \times N \to R^m$$
$$x : R \to R^m$$

Lee 14: 47

Abstracted Structure of the Model of Continuous Dynamics

Between discontinuities, the state trajectory is modeled as a vector function of time,

$$x : T \to R^n \qquad T = [t_0, \infty) \subset R$$



$$\dot{x}(t) = f(x(t),t)$$
$$f : R^m \times T \to R^m$$

The key to the ideal solver semantics is that continuity and local Lipschitz conditions on f are sufficient to ensure uniqueness of the solution over a sufficiently small interval of time.

Lee 14: 48

RK2-3 Solver Approximates Ideal Solver

Given $x(t_n)$ and a time increment h, calculate

$$K_0 = f(x(t_n), t_n) \quad \leftarrow \dot{x}(t_n)$$

$$K_1 = f(x(t_n) + 0.5hK_0, t_n + 0.5h) \quad \leftarrow \text{estimate of } \dot{x}(t_n + 0.5h)$$

$$K_2 = f(x(t_n) + 0.75hK_1, t_n + 0.75h) \quad \leftarrow \text{estimate of } \dot{x}(t_n + 0.75h)$$

then let

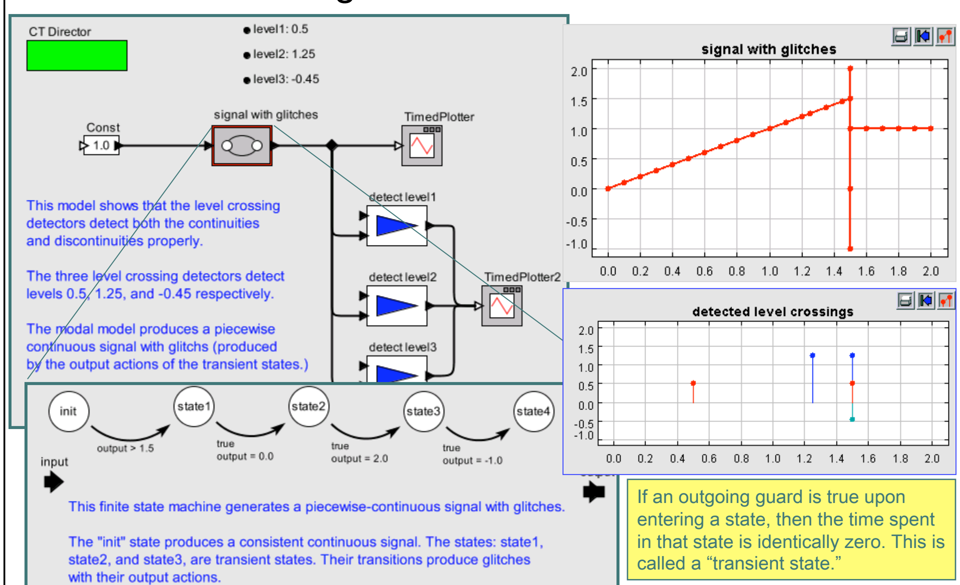
$$t_{n+1} = t_n + h$$

$$x(t_{n+1}) = x(t_n) + (2/9)hK_0 + (3/9)hK_1 + (4/9)hK_2$$

Note that this is strictly (delta) causal, but requires three evaluations of f at three different times with three different inputs.

Lee 14: 49

Generalizing: Multiple Events at the Same Time using Transient States

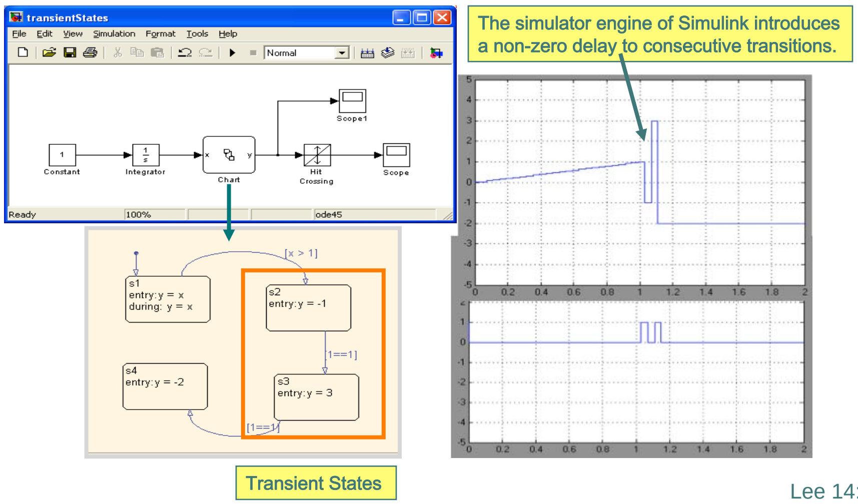


If an outgoing guard is true upon entering a state, then the time spent in that state is identically zero. This is called a “transient state.”

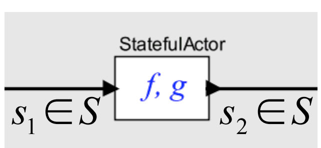
Lee 14: 50

Contrast with Simulink/Stateflow

In Simulink semantics, a signal can only have one value at a given time. Consequently, Simulink introduces solver-dependent behavior.



The Abstract Semantics Supports the Second Requirement as Well



$$S = [T \times N \to R]$$

$$f : \Sigma \times R^m \times T \to R^m$$

$$g : \Sigma \times R^m \times T \to \Sigma$$

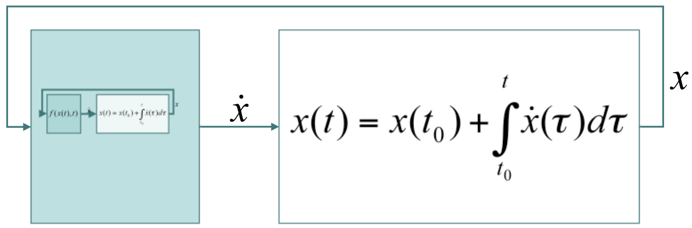
At each $t \in T$ the calculation of the output given the input is separated from the calculation of the new state. Thus, the state does not need to updated until after the step size has been decided upon.

In fact, the variable step size solver relies on this, since any of several integration calculations may result in refinement of the step size because the error is too large.

This deals with the second requirement.

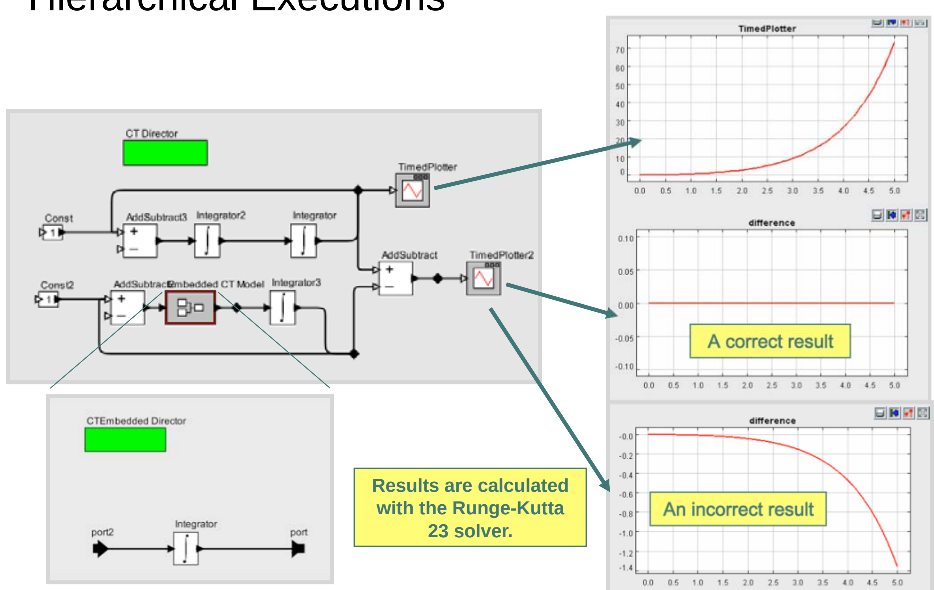
Third Requirement: Compositional Semantics

We require that the system below yield an execution that is identical to a flattened version of the same system. That is, despite having two solvers, it must behave as if it had one.

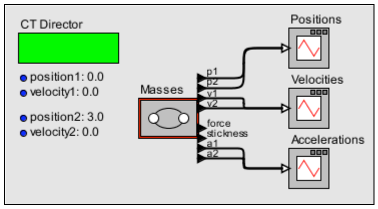


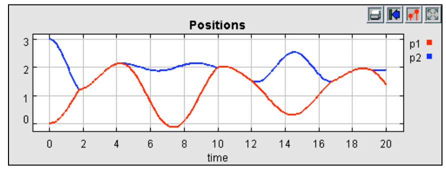
Achieving this appears to require that the two solvers coordinate quite closely. This is challenging when the hierarchy is deeper.

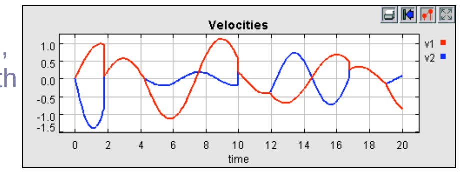
Hierarchical Executions



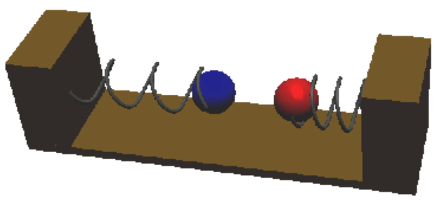
The “right” semantics supports deeper hierarchies

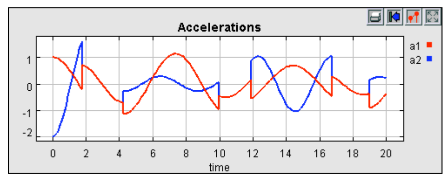




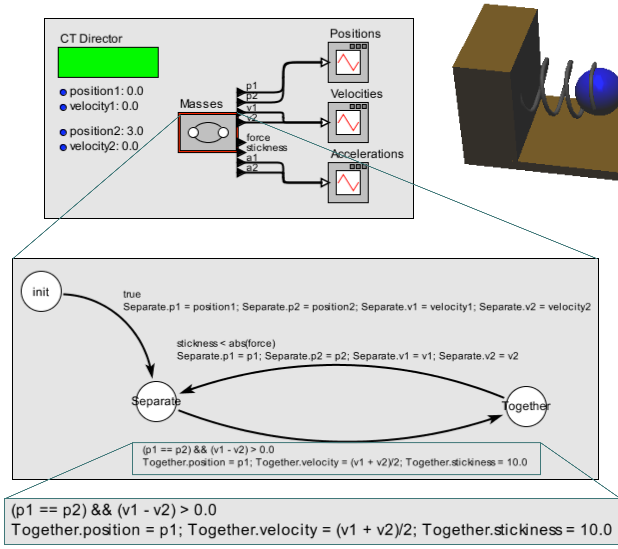


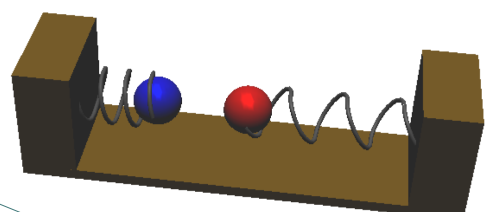
Consider two masses on springs which, when they collide, will stick together with a decaying stickiness until the force of the springs pulls them apart again.





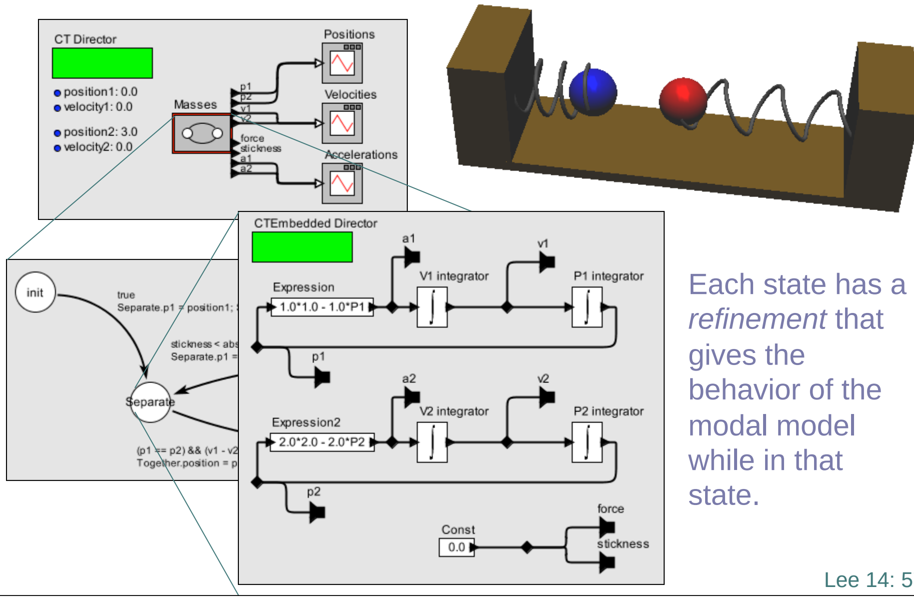
Modal Models





The Masses actor refines to a state machine with two states, *Separate* and *Together*. The transitions have guards and reset maps.

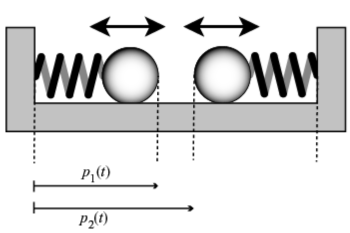
Mode Refinements



Each state has a *refinement* that gives the behavior of the modal model while in that state.

Modeling Dynamics within the Separate Mode

Dynamics while separate:

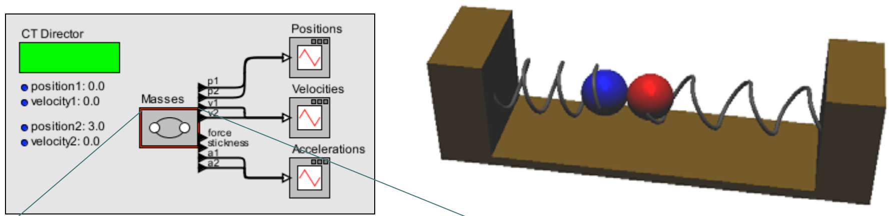


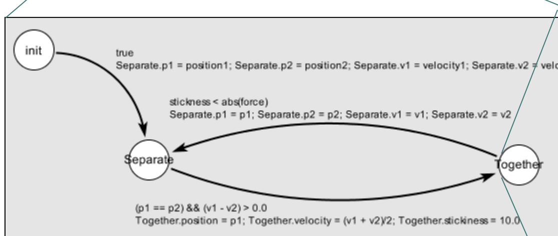
$$\ddot{p}_1(t) = k_1(n_1 - p_1(t))/m_1$$
$$\ddot{p}_2(t) = k_2(n_2 - p_2(t))/m_2.$$

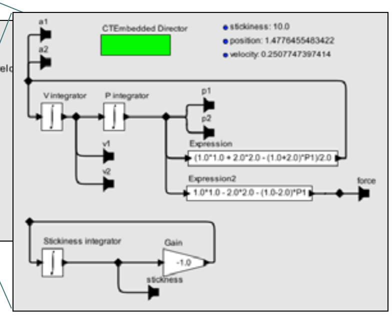
Equivalently:

$$p_1(t) = \int_{t_0}^{t} \left(\int_{t_0}^{\alpha} \frac{k_1}{m_1}(n_1 - p_1(\tau))d\tau + v_1(t_0) \right) d\alpha + p_1(t_0)$$
$$p_2(t) = \int_{t_0}^{t} \left(\int_{t_0}^{\alpha} \frac{k_2}{m_2}(n_2 - p_2(\tau))d\tau + v_2(t_0) \right) d\alpha + p_2(t_0)$$

Mode Refinements (2)



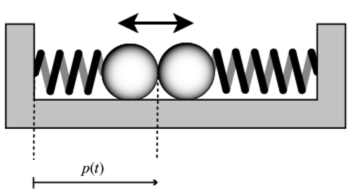




In the Together mode, the dynamics is that of a single mass and two springs.

Modeling Dynamics within the Together Mode

Dynamics while together:

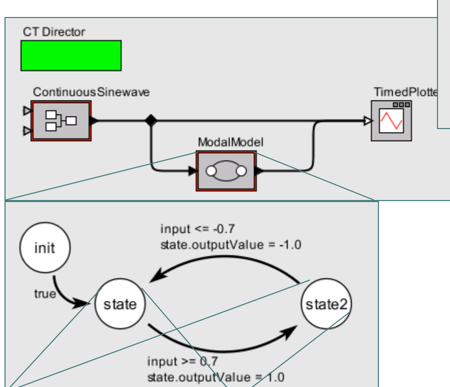


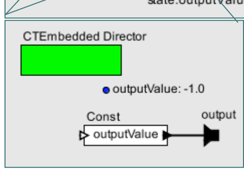
$$\ddot{p}(t) = \frac{k_1 n_1 + k_2 n_2 - (k_1 + k_2) p(t)}{m_1 + m_2}.$$

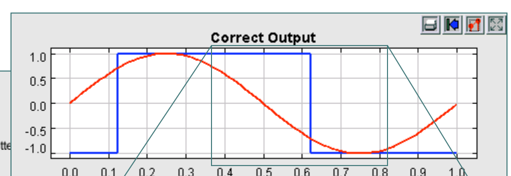
Consider Corner Cases

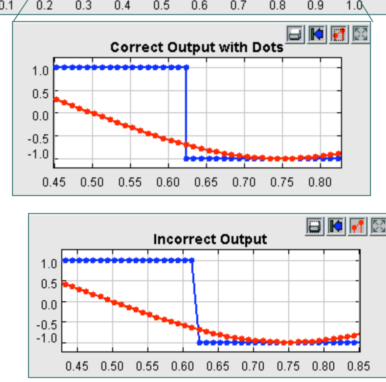
- When triggering transitions based on predicates on discontinuous signals, how should the discontinuity affect the transition?
- What should samples of discontinuous signals be?

Recall Hysteresis Example



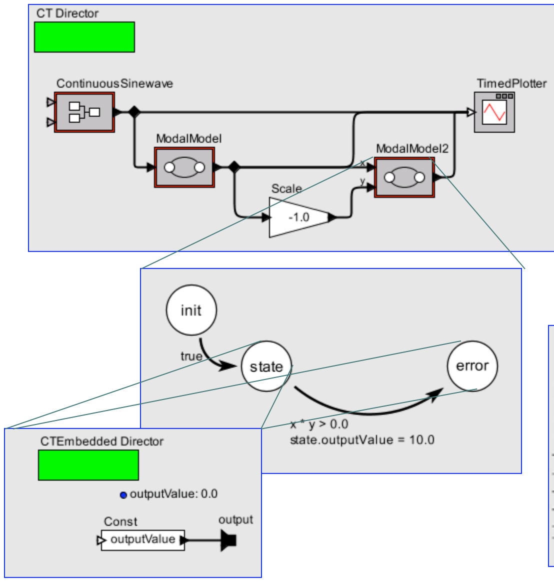




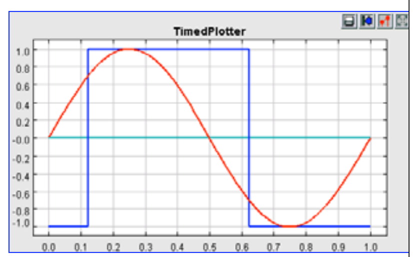


This model generates a discontinuous signal.

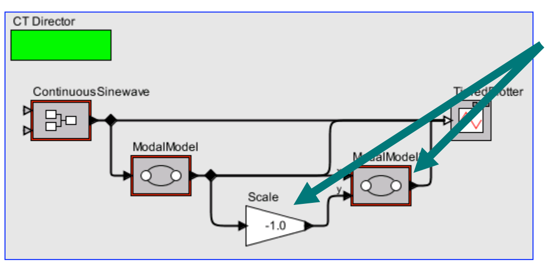
Observing the Discontinuous Signal



ModalModel2 will enter the error state if its inputs ever have the same sign. Note from the plot that it never enters that state (the output would go to 10, but it stays at 0).



Simultaneous Events: The Order of Execution Question



The output of the Scale actor has the same tag as its input, so ModelModel2 sees only two values with opposite signs.

Semantics of a signal:

$$s : T \times N \rightarrow R$$

In Ptolemy II CT, every continuous-time signal has a value at $(t, 0)$ for any $t \in T$. This yields deterministic execution of the above model.

Alternative Interpretations

- *Nondeterministic*: Some hybrid systems languages (e.g. Charon) declare this to be nondeterministic, saying that perfectly zero time delays never occur anyway in physical systems. Hence, ModalModel2 may or may not see the output of ModalModel before Scale gets a chance to negate it.

- *Delta Delays*: Some models (e.g. VHDL) declare that every block has a non-zero delay in the index space. Thus, ModalModel2 will see an event with time duration zero where the inputs have the same sign.

Disadvantages of These Interpretations

- *Nondeterministic*:
  - Constructing deterministic models is extremely difficult
  - What should a simulator do?

- *Delta Delays*:
  - Changes in one part of the model can unexpectedly change behavior elsewhere in the model.

Nondeterministic Ordering

In favor

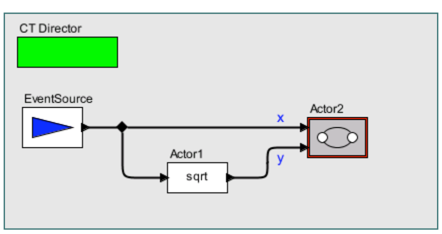
- Physical systems have no true simultaneity
- Simultaneity in a model is artifact
- Nondeterminism reflects this physical reality

Against

- It surprises the designer
  - counters intuition about causality
- It is hard to get determinism
  - determinism is often desired (to get repeatability)
- Getting the desired nondeterminism is easy
  - build on deterministic ordering with nondeterministic FSMs
- Writing simulators that are trustworthy is difficult
  - It is incorrect to just pick one possible behavior!

Lee 14: 67

Consider Nondeterministic Semantics

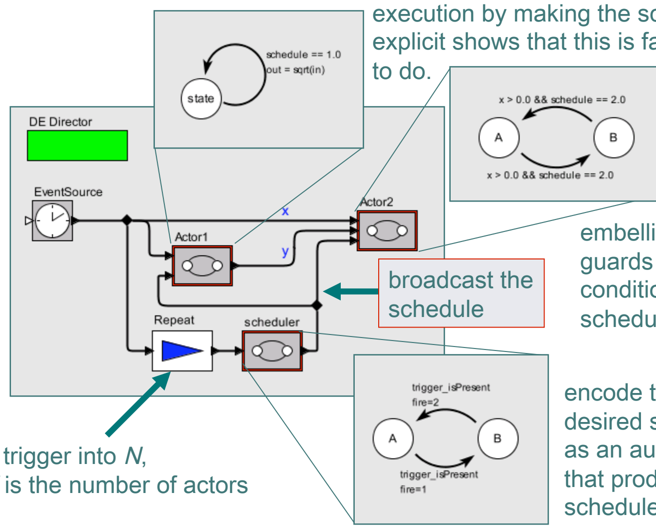


Under nondeterministic semantics, we could modify the model to explicitly schedule the firings.

Suppose we want deterministic behavior in the above (rather simple) model. How could we achieve it?

Lee 14: 68

Non-Deterministic Interaction is the Wrong Answer



An attempt to achieve deterministic execution by making the scheduling explicit shows that this is far too difficult to do.

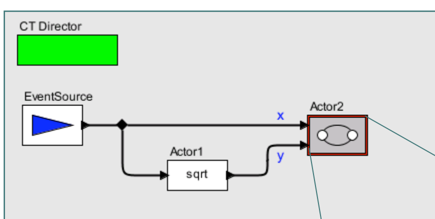
embellish the guards with conditions on the schedule

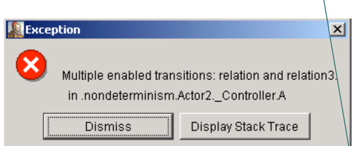
broadcast the schedule

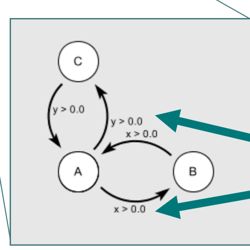
encode the desired sequence as an automaton that produces a schedule

turn one trigger into *N*, where *N* is the number of actors

OTOH: Nondeterminism is Easily Added in a Deterministic Modeling Framework





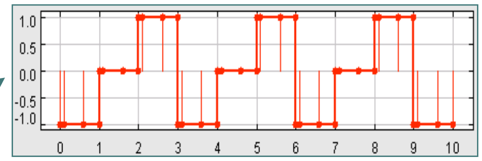


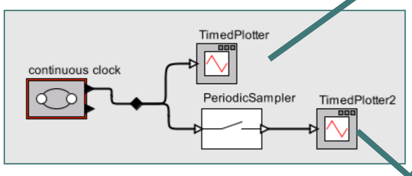
Although this can be done in principle, Ptolemy II CT does not support this sort of nondeterminism by default. What execution trace should it give?

At a time when the event source yields a positive number, both transitions are enabled.

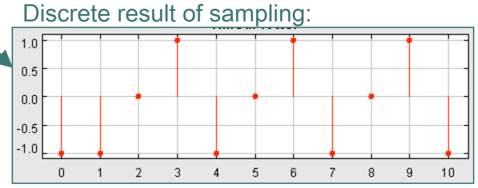
Sampling Discontinuous Signals

Continuous signal with sample times chosen by the solver:





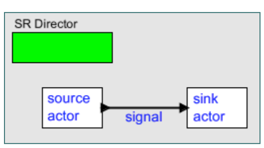
Discrete result of sampling:



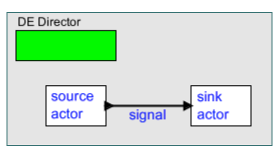
Samples must be deterministically taken at t- or t+. Our choice is t-, inspired by hardware setup times.

Note that in Ptolemy II CT, unlike Simulink, discrete signals have no value except at discrete points.

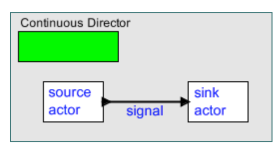
The Continuous (vs. CT) Director
Building continuous-time semantics on SR



A signal has a value or is absent at each tick of a “clock.” By default, all ticks of the “clock” occur at model time 0.0, but they can optionally be spaced in time by setting the *period* parameter of the SR Director.



A signal is a set of events with time stamps (in model time) and the DE Director is responsible for presenting these events in time-stamp order to the destination actor.



A signal is defined everywhere (in model time) and the Continuous Director chooses where it is evaluated. The value of the signal may be “absent,” allowing for signals that are discrete or have gaps.

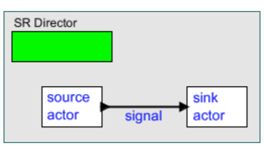
Metric Time in SR

By default, "time" does not advance when executing an SR model in Ptolemy II ("current time" remains at 0.0, a real number).

Optionally, the SR Director can increment time by a fixed amount on each clock tick.

Time in SR Models in Ptolemy II

$$s : T \rightharpoonup V_\varepsilon$$



A signal has a value or is absent at each tick of a "clock." By default, all ticks of the "clock" occur at model time 0.0, but they can optionally be spaced in time by setting the *period* parameter of the SR Director.

Assume the *period* parameter of the SR Director is given by p. The default value is $p = 0$.

- A *tag* is a time-index pair, $\tau = (t, n) \in T = \mathbb{R}_+ \times \mathbb{N}$.
- If $p = 0$, then by default, only the index advances, so actors are fired at model times $(0,0), (0,1), (0,2), \cdots$. Time never advances.
- If $p > 0.0$, then actors are fired at model times $(0,0), (p,0), (2p,0), \cdots$.
  Semantically, signals are "absent" at tags in between.

Execution of an SR Model (Conceptually)

Start with all signals empty (i.e. defined on the empty initial segment).

Initialize all actors.

Invoke the following on all actors until either all signals are defined on the initial segment {(0,0)} or no progress can be made:

if (prefire()) { fire(); }

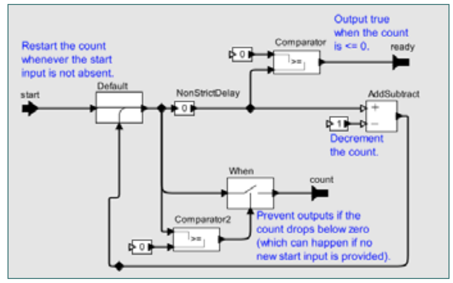
If not all signals are defined on {(0,0)}, declare a causality loop.

Invoke postfire() for all actors.

Choose the next tag *t* ((0,1) or (p, 0))

Repeat to define signals on the initial segment [(0, 0), *t*].

Etc.



The correctness of this is guaranteed by the fixed point semantics. Efficiency, of course, depends on being smart about the order in which actors are invoked.

Metric Time in SR

By default, "time" does not advance when executing an SR model in Ptolemy II ("current time" remains at 0.0, a real number).

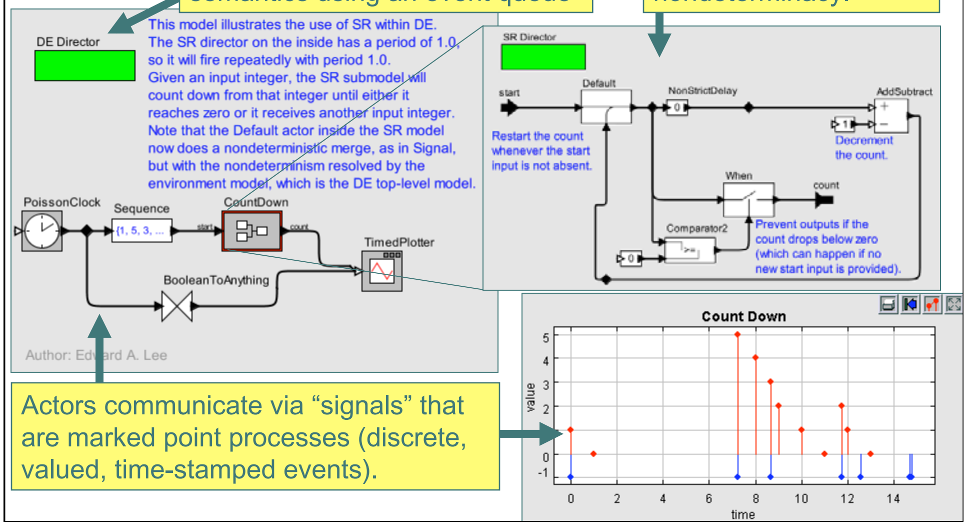
Optionally, the SR Director can increment time by a fixed amount on each clock tick.

More interestingly, SR can be embedded within timed MoCs that model the environment and govern the passage of time.

Discrete Events (DE): A Timed Concurrent Model of Computation

DE Director implements timed semantics using an event queue

SR subsystem implements structured nondeterminacy.



Actors communicate via “signals” that are marked point processes (discrete, valued, time-stamped events).

Advancing Time

A signal is a partial function

$$s : T \rightharpoonup V_\varepsilon$$

defined on an initial segment of

$$T = \mathbb{R}_+ \times \mathbb{N}$$

But how to increment the initial segment on which the signal is defined? It won’t work to just proceed to the next one, as we did with SR.

Execution of a DE Model (Conceptually)

Start with all signals empty.

Initialize all actors (some will post tags on the event ueue).

Take the smallest tag (t, n) from the event queue.

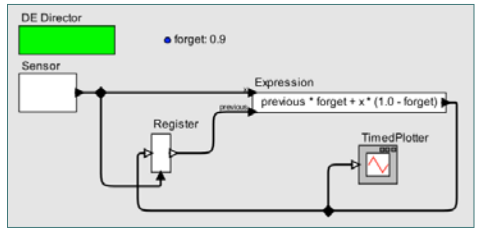
Invoke the following on all actors that have input events until either all signals are defined on the initial segment S = [(0,0), (t,n)] or no progress can be made:

if (prefire()) { fire(); }

If not all signals are defined on S, declare a causality loop.

Invoke postfire() for all actors (some will post tags on the event queue).

Repeat with the next smallest tag on the event queue.



This is exactly the execution policy of SR, except that rather than just choosing the next tag in the tag set, we use a sorted event queue to choose an interval over which to increment the initial segment.

Subtle Difference Between SR and DE

In SR, every actor is fired at every tick of its clock, as determined by a clock calculus and/or structured subclocks.

In DE, an actor is fired at a tag only if it has input events at that tag or it has previously posted an event on the event queue with that tag.

In DE semantics, event counts may matter. If every actor were to be fired at every tick, then adding an actor in one part of a model could change the behavior in another part of the model in unexpected ways.

Recall Subtle Difference Between SR and DE. CT is more like SR.

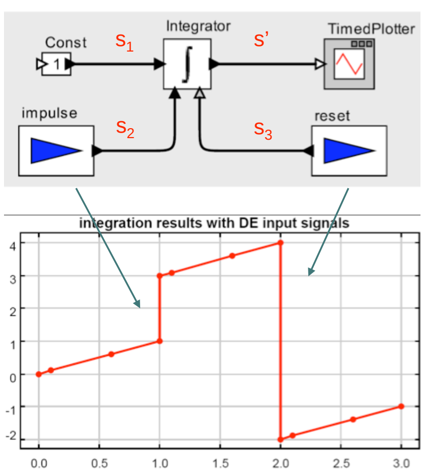
In SR, every actor is fired at every tick of its clock, as determined by a clock calculus and/or structured subclocks.

In DE, an actor is fired at a tag only if it has input events at that tag or it has previously posted an event on the event queue with that tag.

In CT, every actor is fired at every tick of the clock, as determined by an ODE solver.

In CT semantics, a signal has a value at every tag. But the solver to chooses to explicitly represent those values only at certain tags.

Integrator with DE Input Signals



The following table shows the integration results with more complicated DE input signals.

n	0	1	2	3	4	5	6	7
$s_1(1,n)$	1	1	1	1	1	1	1	$\cdots$
$s_2(1,n)$	ε	2	ε	-1	ε	ε	ε	$\cdots$
$s_3(1,n)$	ε	ε	2	ε	3	ε	ε	$\cdots$
$s'(1,n)$	1	3	2	1	3	3	3	$\cdots$
$s_1(1,n)$	1	1	1	1	1	1	1	$\cdots$
$s_2(2,n)$	ε	-1	ε	1	-1	ε	ε	$\cdots$
$s_3(2,n)$	ε	ε	-2	ε	0	ε	ε	$\cdots$
$s'(2,n)$	4	3	-2	-1	0	0	0	$\cdots$

Slide from Haiyang Zheng

Conclusion

- Superdense time is useful for continuous-time models.
- SR provides a foundation for DE and CT.
- Time between “ticks” is chosen in consultation with the solver and breakpoints defined by actors.
- ODE solver can be modeled as an ideal solver semantically.
- Get an operational and denotational semantics that match up to the ability of the solver to match the ideal solver.