# Constructive Models of Discrete and Continuous Physical Phenomena

*Edward A. Lee*

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-15
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-15.html

February 8, 2014

Acknowledgement

This work was supported in part by the iCyPhy Research Center (Industrial Cyber-Physical Systems, supported by IBM and United Technologies), and the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley (supported by the National Science Foundation, NSF awards #0720882 (CSR-EHS: PRET), #1035672 (CPS: Medium: Ptides), and #0931843 (ActionWebs), the Naval Research Laboratory (NRL #N0013-12-1-G015), and the following companies: Bosch, National Instruments, and Toyota).

# Constructive Models of Discrete and Continuous Physical Phenomena

Edward A. Lee

EECS Department, UC Berkeley, Berkeley, CA, USA
eal@eecs.berkeley.edu *

**Abstract.** This paper studies the semantics of models for discrete physical phenomena such as rigid body collisions and switching in electronic circuits. The paper combines generalized functions (specifically the Dirac delta function), superdense time, modal models, and constructive semantics to get a rich, flexible, efficient, and rigorous approach to modeling such systems. It shows that many physical scenarios that have been problematic for modeling techniques manifest as nonconstructive models, and that constructive versions of some of the models properly reflect uncertainty in the behavior of the physical systems that plausibly arise from the principles of quantum mechanics. The paper argues that these modeling difficulties are not reasonably solved by more detailed continuous models of the underlying physical phenomena. Such more detailed models simply shift the uncertainty to other aspects of the model. Since such detailed models come with a high computational cost, there is little justification in using them unless the goal of modeling is specifically to understand these more detailed physical processes. All models in this paper are implemented in the Ptolemy II modeling and simulation environment and made available online.

## 1 The Problem

Many physical phenomena are naturally modeled as being discrete rather than continuous. Modeling and simulating combinations of discrete and continuous dynamics, however, are challenging. Collisions of rigid objects and friction between moving objects are classic examples. Diodes and switches in electrical circuits present similar problems. All known solutions have significant limitations.

The difficulties stem from a number of sources. First, discontinuities make signals non-differentiable, which complicates simulation and analysis. Second, discrete phenomena can cause **chattering** around the discontinuity, where the a solution repeatedly bounces across a discrete boundary. Third, discrete models more easily lead to Zeno conditions than continuous models, where an infinite number of events occur in a finite

* This work was supported in part by the iCyPhy Research Center (Industrial Cyber-Physical Systems, supported by IBM and United Technologies), and the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley (supported by the National Science Foundation, NSF awards #0720882 (CSR-EHS: PRET), #1035672 (CPS: Medium: Ptides), and #0931843 (ActionWebs), the Naval Research Laboratory (NRL #N0013-12-1-G015), and the following companies: Bosch, National Instruments, and Toyota).

time. Finally, and perhaps most importantly, physical phenomena that are most naturally modeled as discrete are among the most poorly behaved and least understood. They frequently exhibit intrinsic nondeterminism and chaotic behaviors.

These sources of difficulty are worth separating. For example, it is not appropriate to condemn a model because it fails to deterministically model an intrinsically nondeterministic physical phenomenon. Nor is it fair to condemn a model for exhibiting Zeno behavior if the Zeno condition manifests outside the regime of parameters for which the model is suited.

Stewart [54] gives an excellent overview of approaches that have been used towards solving these problems for collisions and friction between macroscopic physical objects. In this regime, a solution that admits discrete behaviors can use generalized functions, most commonly the Dirac delta function, Lebesgue integration, measure theory, and differential inclusions. Stewarts argues for embracing discrete behaviors in models, and shows that a well-known paradox in the study of rigid body known as the Painlevé paradox can be resolved by admitting impulsive forces into the model.

A different (and more common) approach is to dispense with discrete models and insist on detailed modeling of the continuous dynamics. Collisions between rigid objects, for example, involve localized plastic deformation, viscous damping in the material, and acoustic wave propagation. Much experimental and theoretical work has been done to refine models of such phenomena, leading to considerable insight into the underlying physical phenomena. We contend, however, that such detailed modeling rarely helps in developing insight about macroscopic system behavior. So when the goal is, for example, to design robotic machinery, it is better to use simpler, more abstract models.

State-of-the-art design and simulation tools, however, do not support simpler models with discrete behaviors well. Modelica [57], for example, is a widely used language with well-supported libraries of models for a large variety of physical systems. Otter, et al. in [46] state that "at the moment, it is not possible to implement the solution with impulses ... in a generic way in Modelica." They offer continuous approximations as an alternative, categorizing three approaches for collisions: impulsive, spring-damper ignoring contact area, and spring-damper including contact area. They describe a library in Modelica that uses the latter two approaches.

Continuous models may indeed more accurately represent the physics, but they come at the price of greatly increased simulation cost and, perhaps more importantly, greatly increased modeling detail. The increased simulation cost is a consequence of the stiffness of the resulting differential equations. The increased modeling detail requires designers to specify much more detail about materials and systems than may be reasonable, particularly at early stages of design. Moreover, such detailed models may just shift the uncertainty from the modeling approximations to the determination of parameters. Is a robot designer able to characterize acoustic propagation in steel for a particular shape of robot arm in a particular range of temperatures and as the product ages? Probably not. So a detailed simulation model based on continuous physical processes may not be any more faithful than a much less detailed model.

In contrast, models that are created for the purpose of providing computer animations, like those described in Erleben et al. [17], are closer to what we need for understanding system dynamics. Computer animation has the very practical driving force that

it must exhibit some behavior in reasonable time, so simulation efficiency is important. In this context, much of the complexity in the models arises from describing motions of complicated shapes in three dimensions. Fortunately, although the techniques this paper can be used for such complex models, there is no need to go beyond simple shapes and one or two-dimensional systems. The simpler contexts are adequate for presenting and analyzing the techniques.

The goal of this paper is improve the trustworthiness of less detailed, more abstract models. The approach is to put the semantics of the models on a solid foundation. If the meaning of a model is absolutely clear, it is much easier to tell whether the model is faithful to the physical system it is modeling, and it is much easier to draw trusted conclusions from simulations of the model.

To provide a solid foundation for abstract models, this paper embraces discrete phenomena modeled using generalized functions, and uses an extended model of time known as **superdense time** to cleanly mix discrete and continuous dynamics. In addition, the technique in this paper supports **modal models**, where a multiplicity of distinct abstract models, each with a well-defined regime of applicability, are combined to model the same system (as in **hybrid systems** [35,1]). Finally, the modeling framework is given a **constructive fixed-point semantics** [9], like that in synchronous-reactive languages [6]. We conjecture that nonconstructive models are suspect on physical grounds, and show that a number of well-known problematic scenarios with modeling discrete physical phenomena result in nonconstructive models. The techniques in this paper have been implemented as a Ptolemy II simulation tool [51], and the models displayed in this paper are all available online at `http://ptolemy.org/constructive/models`.

## 2 Modeling Principles

Golomb, who has written eloquently about the use of models, emphasizes in [20] understanding the distinction between a model and thing being modeled, famously stating "you will never strike oil by drilling through the map." In no way, however, does this diminish the value of a map! Models are created to gain insight into the thing being modeled and to predict behaviors that have not yet been observed. Although precision may be helpful in models, providing insight is more important than being precise. And insight follows more easily from simple models than from complex ones. This point is supported by [11], who state "essentially, all models are wrong, but some are useful." Of course the usefulness of a model depends on its **fidelity**, the degree to which it accurately imitates the system being modeled. But models are *always* an approximation.

When choosing models, it is essential to bear in mind what sort of insight we are seeking. For collisions of rigid objects, for example, we would not choose the same model if the goal is to understand the physics of collisions as the one we would choose if the goal is to design a robotic factory floor, where machinery comes into contact. In this paper, we assume that the goal is not to understand the physics. Such a goal would automatically bias the choice towards more detailed models. Instead, we assume the goal is to understand how collisions affect a *system* behavior. This biases the choice towards simpler models that can be simulated efficiently.

A model with high fidelity has high fidelity only within some regime of operation. For example, a digital logic model of an integrated circuit is not valid if the circuit is melting. One way to construct a high-fidelity model that is simple is to cover a more limited regime, i.e. to make more assumptions about the operating conditions. This creates a need for multiple models, each covering a distinct regime. We call a regime of operation, together with its assumptions about the operating conditions, a **mode**. A **modal model** is a collection of such mode models together with a logic for switching discretely between mode models. Hybrid systems [35] are well known examples of modal models. Modal models require a modeling framework that embraces discrete events in order to support discrete mode switches.

For models to be useful, their meaning must be clear. A map with mysterious, undefined symbols is not as useful as one with a legend. For this reason, we put considerable emphasis on **model semantics**. In this paper, we are interested in models for simulation, and we treat simulation as an execution of the model. A model, therefore, is a program, the modeling language is a programming language, and a simulation is an execution of the program. A programming language is more useful if there is a very clear definition of what it means to **correctly** execute the program. There has to be a well-defined "right answer," and any execution that does not yield that right answer is an incorrect execution. There could even be more than one "right answer," in which case the program is **nondeterminate**, but in this case, the *set* of right answers must be well defined.

When simulating physical systems, this immediately brings us to the problem that the continuums of physical dynamics cannot be exactly emulated by a computation. Instead, we use numerical methods to approximately solve ordinary differential equations (ODEs). Since there are many techniques, all of which are approximate, it seems we have to give up on the notion of a "right answer."

But we do not. First, the discrete portions of a computation, such as mode switches an impulsive events, are well matched to what computers can do, and hence we can easily assign them a computational semantics. Since those are the focus of this paper, we are on solid ground. But even the continuous dynamics between discrete events can often be treated as a computational problem with a well defined "right answer." Specifically, we can adopt the **ideal solver semantics** of [34], which points out that under certain conditions on a system of ODEs (Lipschitz continuity), a uniquely defined solution exists over a non-zero time interval. This uniquely defined solution gives a benchmark against which the correctness of computational answer can be determined (and in certain cases, even admits for exact answers through symbolic solutions). This is analogous to the use of real numbers as a benchmark against which the correctness of floating-point arithmetic is assessed.

In summary, models are maps, not oil fields. The meaning of the map needs to be clear. Faithfulness to the oil field needs to be good enough that the intended purpose of the map can be met. Our models, unlike a map, are executable, and we need for the execution to be efficient, and for there to be a clear distinction between correct and incorrect executions.

# 3 Time

Time is central to our approach to modeling. We require a model of time that combines a time continuum, over which physical dynamics can evolve, and discrete events, modeling abrupt changes in state of the system. In this section, we review the superdense model of time, the notion of discreteness, and the notion of piecewise continuity, which is essential for our models to work well with practical ordinary differential equation (ODE) solvers.

## 3.1 Superdense Time

We use a model of time known as **superdense time** [38,35,31,13]. A superdense time value is a pair $(t,n)$, called a **time stamp**, where $t$ is the **model time** and $n$ is an **index** (also called a **microstep**). The model time represents the time at which some event occurs, and the microstep represents the sequencing of events that occur at the same model time. Two time stamps $(t,n_1)$ and $(t,n_2)$ can be interpreted as being **simultaneous** (in a weak sense) even if $n_1 \neq n_2$. Strong simultaneity requires the time stamps to be equal (both in model time and microstep).

To understand the role of the microstep, consider Newton's cradle, a toy with five steel balls suspended by strings. If you lift the first ball and release it, it strikes the second ball, which does not move. Instead, the fifth ball reacts by rising. Consider the momentum $p$ of the second ball as a function of time. The second ball does not move, so its momentum must be everywhere zero. But the momentum of the first ball is somehow transferred to the fifth ball, passing through the second ball. So the momentum cannot be always zero.

Let $\mathbb{R}$ represent the real numbers. Let $p\colon \mathbb{R} \to \mathbb{R}$ be a function that represents the momentum of this second ball, and let $\tau$ be the time of the collision. Then

$$p(t) = \begin{cases} P & \text{if } t = \tau \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

for some constant $P$ and for all $t \in \mathbb{R}$. Before and after the instant of time $\tau$, the momentum of the ball is zero, but at time $\tau$, it is not zero. Momentum is proportional to velocity, so

$$p(t) = Mv(t),$$

where $M$ is the mass of the ball. Hence, combining with (1),

$$v(t) = \begin{cases} P/M & \text{if } t = \tau \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The position of a mass is the integral of its velocity,

$$x(t) = x(0) + \int_0^t v(\tau)d\tau,$$

where $x(0)$ is the initial position. The integral of the function given by (2) is zero at all $t$, so the ball does not move, despite having a non-zero momentum at an instant.

The above physical model mostly works to describes the physics, but it has two flaws. First, it violates the basic physical principle of conservation of momentum. At the time of the collision, all three middle balls will simultaneously have non-zero momentum, so seemingly, aggregate momentum has magically increased.

Second, the model cannot be directly converted into a discrete representation (see Section 3.3 below). A discrete representation of a signal is a sequence of values that are ordered in time. Any such representation of the momentum in (1) or velocity in (2) is ambiguous. If the sequence does not include the value at the time of the collision, then the representation does not capture the fact that momentum is transferred through the ball. If the representation does include the value at the time of the collision, then the representation is indistinguishable from a representation of a signal that has a non-zero momentum over some interval of time, and therefore models a ball that does move. In such a discrete representation, there is no semantic distinction between an instantaneous event and a rapidly varying continuous event.

Superdense time solves both problems. Specifically, the momentum of the second ball can be unambiguously represented by a sequence of samples where $p(\tau,0) = 0$, $p(\tau,1) = P$, and $p(\tau,2) = 0$, where $\tau$ is the time of the collision. The third ball has non-zero momentum only at superdense time $(\tau,2)$. At the time of the collision, each ball first has zero momentum, then non-zero, then zero again, all in an instant. The event of having non-zero momentum is weakly simultaneous for all three middle balls, but not strongly simultaneous. Momentum is conserved, and the model is unambiguously discrete.

One could argue that the physical system is not actually discrete. Even well-made steel balls will compress, so the collision is actually a continuous process, not a discrete event. This may be true, but when building models, we do not want the modeling formalism to force us to construct models that are more detailed than is appropriate. Such a model of Newton's cradle would be far more sophisticated, and the resulting non-linear dynamics would be far more difficult to analyze. The fidelity of the model may improve, but at a steep price in understandability and analyzability. Moreover, if the properties of the material and the dynamics of the collision are not well understood, the fidelity of the model may actually degrade as more detail is added.

The Newton's cradle example shows that physical processes that include instantaneous events are better modeled using functions of the form $p\colon \mathbb{R}\times\mathbb{N}\to\mathbb{R}$, where $\mathbb{N}$ represents the natural numbers, rather than the more conventional $p\colon \mathbb{R}\to\mathbb{R}$. The latter is adequate for continuous processes, but not for discrete events. At any time $t\in\mathbb{R}$, the signal $p$ has a sequence of values, ordered by their microsteps. This signal cannot be misinterpreted as a rapidly varying continuous signal.

Superdense time is ordered lexicographically (like a dictionary), which means that $(t_1,n_1) < (t_2,n_2)$ if either $t_1 < t_2$, or $t_1 = t_2$ and $n_1 < n_2$. Time stamps are a particular realization of **tags** in the tagged-signal model of [27].

### 3.2 Piecewise Continuity

So that we can leverage standard, well-understood numerical integration methods, we require signals to be piecewise-continuous in a specific technical sense. Consider a real-valued superdense-time signal $x\colon \mathbb{R}\times\mathbb{N}\to\mathbb{R}$. At each real-time $t\in\mathbb{R}$, we require that

$x(t,n)$ settle to a final value and stay there. Specifically, we require that for all $t \in \mathbb{R}$, there exist an $m \in \mathbb{N}$ such that

$$\forall n > m, \quad x(t,n) = x(t,m). \tag{3}$$

A violation of this constraint is called a **chattering Zeno** condition. The value of the signal changes infinitely often at a model time $t$. Such conditions would prevent an execution from progressing beyond real time $t$, assuming the execution is constrained to produce all values in chronological order.

Assuming $x$ has no chattering Zeno condition, then there is a least value of $m$ satisfying (3). We call this value of $m$ the **final microstep** and $x(t,m)$ the **final value** of $x$ at $t$. We call $x(t,0)$ the **initial value** at time $t$. If $m = 0$, then $x$ has only one value at time $t$.

Define the **initial value function** $x_i : \mathbb{R} \to \mathbb{R}$ by

$$\forall\, t \in \mathbb{R}, \quad x_i(t) = x(t,0).$$

Define the **final value function** $x_f : \mathbb{R} \to \mathbb{R}$ by

$$\forall\, t \in \mathbb{R}, \quad x_f(t) = x(t,m_t),$$

where $m_t$ is the final microstep at time $t$. Note that $x_i$ and $x_f$ are conventional continuous-time functions. A **piecewise continuous** signal is defined to be a function $x$ of the form $x : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$ with no chattering Zeno conditions that satisfies three requirements:

1. the initial value function $x_i$ is continuous on the left at all $t \in \mathbb{R}$;
2. the final value function $x_f$ is continuous on the right at all $t \in \mathbb{R}$; and
3. $x$ has only one value at all $t \in \mathbb{R} \backslash D$, where $D$ is a discrete subset of $\mathbb{R}$.

The last requirement is a subtle one that deserves further discussion. First, the notation $\mathbb{R} \backslash D$ refers to a set that contains all elements of the set $\mathbb{R}$ except those in the set $D$. $D$ is constrained to be a discrete set, as defined below. Intuitively, $D$ is a set of time values that can be counted in temporal order. It is easy to see that if $D = \emptyset$ (the empty set), then $x_i = x_f$, and both $x_i$ and $x_f$ are continuous functions. Otherwise each of these functions is piecewise continuous.

Such piecewise-continuous signals coexist nicely with standard ODE solvers. At the time of a discontinuity or discrete event, the final value signal provides the initial boundary condition for the solver. The solver then works with an ordinary continuous signal until the time of the next discontinuity or discrete event, and the solver provides the initial value of the signal at the time of that next event.

Techniques for constructing models that produce only piecewise-continuous signals are described in [12]. For our purposes here, it is sufficient to note that modal models, as defined below, always produce piecewise-continuous signals.

### 3.3 Discreteness

A set $D$ is a **discrete set** if it is a totally ordered set (for any two elements $d_1$ and $d_2$, either $d_1 \leq d_2$ or $d_1 > d_2$), and there exists a one-to-one function $f : D \to \mathbb{N}$ that is **order preserving**. Order preserving simply means that for all $d_1, d_2 \in D$ where $d_1 \leq d_2$,

we have that $f(d_1) \leq f(d_2)$. The existence of such a one-to-one function ensures that we can arrange the elements of $D$ *in temporal order*. Notice that $D$ is a countable set, but not all countable sets are discrete. For example, the set $\mathbb{Q}$ of rational numbers is countable but not discrete. There is no such one-to-one function.

In order to be able to cleanly mix discrete and continuous behaviors, we introduce the idea that a signal can be **absent** at a superdense time $(t,n)$. We write this as

$$x(t,n) = \varepsilon.$$

A conventional continuous-time signal is nowhere absent. Discrete-event signals, by contrast, are absent almost everywhere.

A **discrete-event** signal is a function from superdense time to some value set that includes $\varepsilon$, where the signal is non-absent only at a discrete subset of times. I.e., a discrete-event signal is absent almost everywhere, and the superdense times at which it is not absent form a discrete set. An **event** in a discrete-event signal is a time-value pair $((t,n),v)$, where $(t,n)$ is a superdense time and $v$ is a non-absent value. A discrete-event signal has a first event, a second event, etc., i.e. an ordered countable set of events.

The concept of piecewise continuity can be extended to discrete-event signals. A discrete-event signal is a function

$$x\colon \mathbb{R} \times \mathbb{N} \to \{\varepsilon\} \cup U,$$

where $U$ is some value set, and $x(t,n) = \varepsilon$ for all $(t,n) \in (\mathbb{R} \times \mathbb{N}) \setminus D$, where $D$ is a discrete set. That is, the signal is absent almost everywhere, and is present only at a discrete subset of superdense times. A **piecewise-continuous discrete-event signal** is defined as a discrete-event signal whose initial value and final value functions always yield absent,

$$\forall t \in \mathbb{R}, \quad x_i(t) = \varepsilon, \quad x_f(t) = \varepsilon.$$

Such signals can coexist easily with numerical ODE solvers, since the signals seen by the solver, which are initial and final value signals, are simply absent. The solver ignores them.

The above definitions are used in [25]. Benveniste et al. in [7] define "discrete" differently to mean that "each instant has unique previous and next instants." This is a much weaker definition than ours here. We prefer our definition, because every event in a discrete-event signal has a finite number of predecessor events in the signal. This property is essential to being able to compute the events in a signal.

We define a **continuous-time signal** to be a signal whose value is not absent at any superdense time. Clearly, a continuous-time signal is not a discrete-event signal. But we can have signals that are neither continuous-time signals nor discrete-event signals. This becomes important with modal models, where a signal may be present in some modes and not in others. We will require such signals to be piecewise continuous.

Any signal that is not a discrete-event signal will need to either be represented symbolically or numerically approximated in any computation. This is because such a signal has an uncountably infinite number of values, and no computational system can directly represent such sets of values. Standard ODE solvers produce estimated **samples** of continuous-time signals. The time spacing between samples is determined by the step-size control of the solver. The samples themselves are defined on a discrete subset of superdense time.

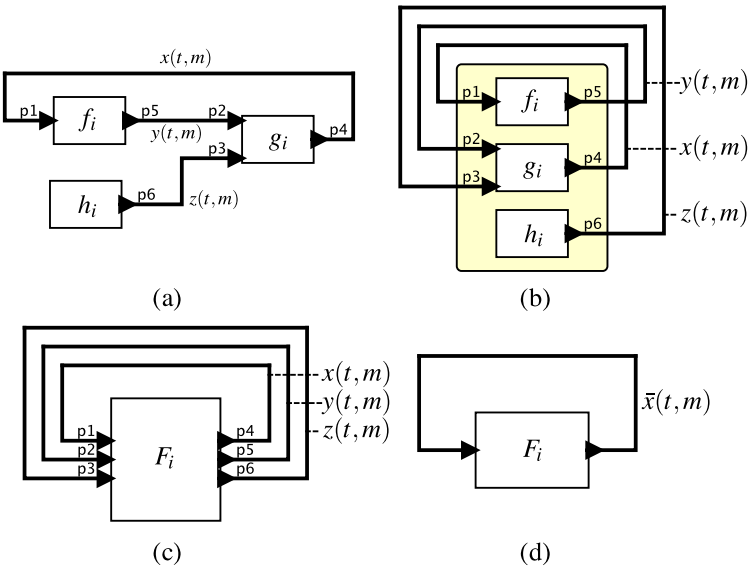


**Fig. 1.** A composition of actors with a constructive fixed-point semantics.

### 3.4 An Alternative Model of Time

Note that an alternative model of time that can accomplish the same objectives as superdense time is studied in [7,42,10]. Their construction is based on nonstandard analysis [33], which, similarly to superdense time, has an infinite number of points at every real time point. These points are represented as convergent sequences, and a total order is induced over these sequences by means of a measure-theoretic construction. It has the property that every non-standard time has an immediate predecessor and an immediate successor, which the authors say provides an operational semantics. However, while an operational semantics does require the notion of a discrete step of computation, it also requires that the number of steps preceding any given step be finite. That is not automatically provided by the nonstandard semantics, and when it is provided, the solutions seem to be isomorphic with our much simpler superdense time construction. Hence, it does not appear to this author that anything is gained by going to a more complex mathematical formulation.

## 4 Constructive Fixed-Point Semantics

The next essential element in our rigorous modeling framework is the **constructive fixed-point semantics** [9], which defines the meaning of a model as a composition of components. The semantics we choose for hybrid systems is that of [32], which is implemented in Ptolemy II [12]. We review the approach here.

A model is assumed to be a graph of **actors**, as shown in Figure 1. An execution of the model (a simulation) will choose a discrete subset $D \subset \mathbb{R} \times \mathbb{N}$ of superdense time values at which to evaluate the model. The elements of $D$ will be selected in order by a solver, beginning at some start time, say $(0,0)$. At each $(t,m) \in D$, the solver will find a value for each of the signals in the model, using a constructive procedure described below. For example, at the start time $(0,0)$, the solver will find the values $x(0,0), y(0,0)$, and $z(0,0)$ in Figure 1.

After finding the values of all signals at a time $(t,m) \in D$, the solver will choose the next time $(t',m')$ at which to evaluate the model. To do this, it must first ensure that it has found the *final value* of each signal. If it has not, then it will increment only the microstep, so $(t',m') = (t,m+1)$. If it has found the final value of all signals, then it will consult an **event queue**, which contains a record of future discrete events, and a **numerical ODE solver**, which determines a step size $\Delta$ that achieves a desired numerical accuracy. It then chooses the lesser of $(t+\Delta,0)$ and $(t_n,m)$, where $(t_n,m)$ is the superdense time of the earliest event in the event queue.

Each superdense time in $D$ is called a **tick**. The set $D$ of ticks is discrete. The ticks can thus indexed by the natural numbers, so that

$$D = \{\tau_0, \tau_1, \cdots, \tau_i, \cdots\}, \tag{4}$$

where $i < j \in \mathbb{N}$ implies that $\tau_i < \tau_j$.

At each $(t,m) \in D$, the solver needs to find the value of each signal. To support this, each actor provides a function from its input values to its output values. There are some technical constraints on these functions, so we must be careful defining them.

Consider the actor labeled $f_i$ in Figure 1. It has one input port p1 and one output port p5. Assume these ports have data type $V_1$ and $V_5$, respectively, where $V_i$ is the set of values that a signal at port $i$ may take on. If the actor can tolerate an absent input, then $\varepsilon \in V_1$, where $\varepsilon$ represents "absent." If the actor may produce no output, then $\varepsilon \in V_5$. Define an **extended data type** for the $k$-th signal as a set

$$\tilde{V}_k = \{\perp\} \cup V_k.$$

That is, $\tilde{V}_k$ is just the data type, augmented with an additional element, $\perp$, pronounced "bottom," which represents **unknown**. The actor function has these augmented sets as domain and codomain. So the functions in Figure 1 have the form

$$\begin{aligned} f_i &: \tilde{V}_1 \to \tilde{V}_5 \\ g_i &: \tilde{V}_2 \times \tilde{V}_3 \to \tilde{V}_4 \\ h_i &: \{\emptyset\} \to \tilde{V}_6. \end{aligned}$$

The subscript $i$ is the tick index, matching the subscripts in (4). On the first tick, for example, the actor labeled $f_i$ provides a function $f_0$. On the second tick, it provides a (possibly different) function $f_1$.

The function $h_i$ deserves some comment. The actor has no input port, so the domain of its function $h_i$ is a **singleton set**, a set with only one element. The function, therefore, specifies exactly one element in its codomain, $\tilde{V}_6$. At each tick $i$, the actor produces a fixed output given by $h_i(\emptyset)$.

Defining an actor to operate on the extended data type is not an onerous burden. In fact, a simple default can be provided, where $f(\perp) = \perp$. An actor that requires all its inputs to be known in order to be able to produce a known output is called a **strict actor**. An actor that produce a known output without all its inputs being known is a **nonstrict actor**.

As illustrated in Figure 1, a network of actors can be rearranged to define a single function $F_i$ that operates on all the signal values at tick $i$ and yields all the signal values at tick $i$. The task of the solver, therefore, is to find the signal values $\bar{x}(t,m)$ that are a **fixed point** of the function $F_i$, i.e. that satisfy

$$F_i(\bar{x}(t,m)) = \bar{x}(t,m),$$

where $\bar{x}$ is a three-tuple of signal values. The fixed point is the **meaning** (semantics) of the model at tick $i$.

Three key questions now arise. Does a fixed point exist? Is it unique? How should the solver find it? These questions are answered by the **constructive** fixed-point semantics.

Recall that the augmented sets $\tilde{V}_i$ include an element $\perp$ called unknown. The constructive procedure for finding a fixed point is to start at each tick with every signal value being $\perp$, and then to evaluate the actor functions *in any order* until a fixed point is found. With a simple and easily realized constraint on each function (monotonicity, explained below), this procedure always yields a fixed point, and the fixed point is uniquely defined to be the least fixed point in a particular partial order that we will define.

Before giving the formalism, we consider an example to develop intuition. Suppose that in Figure 1 all the data types are $V_i = \mathbb{R}$ (approximated by floating-point numbers). Suppose that the actor functions are such that $f_i$ increments its input by 1.0, $g_i$ sums its two inputs, and $h_i$ produces 1.0. Then what is the fixed point?

We initialize all signal values to $\perp$ and evaluate functions in any order. Suppose we start with $f_i$. Its input is unknown, so its output is unknown. We cannot increment a value by 1.0 if we do not know what the value is. The actor is strict. Suppose we next evaluate $g_i$. Again, its inputs are unknown, so its output is unknown. It too is strict. Suppose we next evaluate $h_i$. Its output is 1.0. This provides more information for $g_i$, so we should evaluate that function again. But its output remains unknown because the input at port p2 is unknown. We have a reached a fixed point where

$$\bar{x}(t,m) = (x,y,z) = (\perp,\perp,1.0).$$

When the fixed point that is found by this procedure yields one or more unknown signals, the model is **nonconstructive**. In this case, it is nonconstructive because there is a **causality loop**, where the outputs of $f_i$ and $g_i$ depend directly on their inputs, and each input equals the other's output.

Suppose instead that at the first tick, $f_0$ produces an initial value, say 0.0, and $f_i$ for each $i \geq 1$ produces the input received in the previous tick, $i-1$.[1] In this case, the first

[1] This actor implements the "**fby**" (followed by) function of the Lustre synchronous language [21].

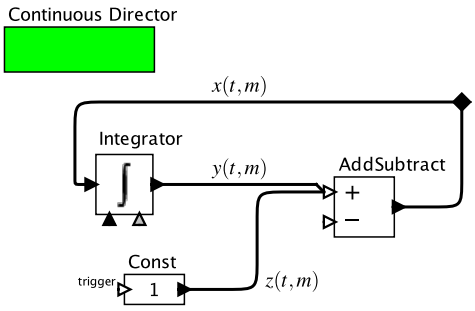


**Fig. 2.** An executable model with the structure of Figure 1. [online]

fixed point found by the above procedure will yield

$$\bar{x}(0,0) = (x,y,z) = (1,0,1).$$

This is not the final value at model time 0, so the solver should choose $(0,1)$ as the next tick. At that tick, the fixed point will be

$$\bar{x}(0,1) = (1,1,2).$$

In the next tick,

$$\bar{x}(0,2) = (1,2,3).$$

With these actor definitions, there is no final value at model time 0, so the model exhibits chattering Zeno behavior.

Suppose instead that $f_i$ implements an integration function. An actual Ptolemy II model with this structure is shown in Figure 2. The model includes an instance of the Continuous director, which implements the semantics described in this paper. An execution of the model is shown in Figure 3. The horizontal axis of this plot shows the real coordinate only (the model time) of superdense time. In this case, none of the signals is discontinuous, so every signal has its final value at microstep zero. Moreover, every signal is a continuous-time signal (it is not absent anywhere). The points on the horizontal axis where values are provided have been chosen by the numerical ODE solver, which is selected as a parameter of the Continuous director. In this case, it is a variable-step-size Runge-Kutta 2-3 solver.

The model in Figure 2 is constructive because the Integrator actor does not need to know its input in order to produce an output. A fact about integration is that for any real-valued function $x\colon \mathbb{R} \to \mathbb{R}$, if we define

$$y(t) = \int_0^t x(\tau) d\tau,$$

then $y(t)$ does not depend on $x(t)$. It *does* depend on the values $x(\tau)$ for all $0 \leq \tau < t$. But it does not depend on the value at $t$. The integrator is nonstrict. The Integrator actor

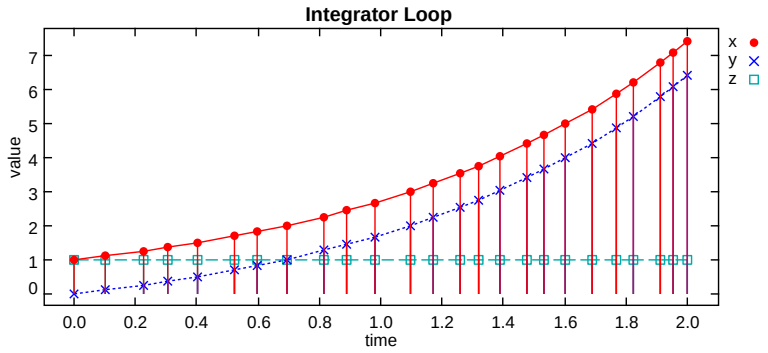


**Fig. 3.** An execution of the model in Figure 2.

provides a numerical approximation to this function $y$, assuming the input is $x$. Hence, if at a tick in superdense time, the input $x(t,m) = \bot$, then the actor can nevertheless provide an output $y(t,m)$. So the causality loop is broken.[2]

We can now outline why this procedure works. The actor functions are required to satisfy a particular technical constraint called **monotonicity**. Specifically, for each set $\tilde{V}_i$, we define a **flat partial order**, where $\bot < v$ for all $v \in V_i$, and all $v \in V_i$ are incomparable with each other. For a set $\tilde{V}_i \times \tilde{V}_j$, as is needed for the domain of $g_i$ in Figure 1 for example, we define a pointwise partial order from this flat partial order. That is, for any $(v,w),(x,y) \in \tilde{V}_i \times \tilde{V}_j$,

$$(v,w) \leq (x,y) \Leftrightarrow v \leq x \wedge w \leq y.$$

In this partial order, we require every actor function to be monotonic. Specifically, a function $f_i : \tilde{V}_1 \to \tilde{V}_5$ is monotonic if for all $x,u \in \tilde{V}_1$,

$$x \leq u \Rightarrow f_i(x) \leq f_i(u).$$

In the flat partial order, this is not an onerous restriction on the actor function $f_i$. Specifically, it just says that if $f_i(\bot)$ yields some value $y$, then $f_i(v) = y$ for any $v \in \tilde{V}_1$. In words, if the actor function can produce an output when its input is unknown, then it should produce that same output for any known input. This is intuitive, and any violation of this principle would mean that the function actually *did* need to know the input.

If all actor functions in a network are monotonic in this sense, then the well known Kleene fixed-point theorem [15] guarantees that there is a unique least fixed point and that the procedure we have outlined above will find that unique fixed point in a finite number of steps.

[2] Note that an **implicit solver** would, in fact, create a direct dependence on the input $x$. It is common when using implicit solvers in such feedback loops to extrapolate to estimate the input $x$. Such extrapolation again breaks the causality loop.

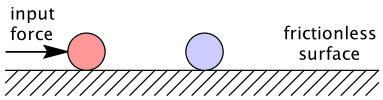


**Fig. 4.** Two balls on a frictionless surface.

It is important to note that for our purposes here it is not important what scheduling strategy is used to find the fixed point. We have stated that functions can be invoked *in any order*, but clearly some choices of orderings will be more efficient than others. See [16] for a discussion of optimized execution.

One final loose end remains. We have stated that actor functions, such as $f_i$ in Figure 1, are indexed by the tick $i$. This means that an actor can realize a new function in each tick. How does it progress from one function to the next? Conceptually, each actor is a state machine, and the function $f_i$ is associated with a state of the actor. The actor changes state when moving from one tick to the next, *after* the constructive fixed-point procedure has concluded. Thus, when the actor determines its next state, assuming the model is constructive, all inputs are known. This strategy is called the **actor abstract semantics** (see [26] and [51]). In the case of the Integrator actor, the progression from one state to the next is simply the integration algorithm. The result of that progression is the new value of the state of the integrator, which can then be produced in the next tick as an output without knowing the current input at that tick.

## 5 Collisions

We now consider the first of the families of discrete physical phenomena that we will consider in this paper, collisions between rigid objects. Consider two objects with masses $m_1$ and $m_2$ colliding on a one-dimensional frictionless surface. We would like to treat the collision as an instantaneous event and are interested in determining the velocity after a collision. Newton's laws of motion imply that total momentum is conserved. If the velocities of the masses before the collision are $v_1$ and $v_2$, and after the collision are $v_1'$ and $v_2'$, then conservation of momentum requires that

$$m_1 v_1' + m_2 v_2' = m_1 v_1 + m_2 v_2. \tag{5}$$

For notational simplicity, we leave off the dependence on time of the velocities, for now. Consider first perfectly elastic collisions, where no kinetic energy is lost. Conservation of kinetic energy requires that

$$\frac{m_1 (v_1')^2}{2} + \frac{m_2 (v_2')^2}{2} = \frac{m_1 (v_1)^2}{2} + \frac{m_2 (v_2)^2}{2}. \tag{6}$$

We have two equations and two unknowns, $v_1'$ and $v_2'$. Because of the quadratic, there are two solutions to these equations. The trivial solution represents the absence of a

collision, where $v_1' = v_1$ and $v_2' = v_2$. The second solution is

$$v_1' = \frac{v_1(m_1 - m_2) + 2m_2v_2}{m_1 + m_2} \tag{7}$$

$$v_2' = \frac{v_2(m_2 - m_1) + 2m_1v_1}{m_1 + m_2}. \tag{8}$$

Note that if $m_1 = m_2$, then the two masses simply exchange velocities.

In practice, most collisions of macroscopic physical objects lose kinetic energy. A common way to model this is to use an empirical quantity called the **coefficient of restitution**, denoted $e$ and defined to be the relative speed after a collision divided by the relative speed before the collision. Using such a coefficient, the velocities after the collision are given by [5]

$$v_1' = \frac{em_2(v_2 - v_1) + m_1v_1 + m_2v_2}{m_1 + m_2} \tag{9}$$

$$v_2' = \frac{em_1(v_1 - v_2) + m_1v_1 + m_2v_2}{m_1 + m_2}. \tag{10}$$

The coefficient of restitution is determined experimentally for a particular pair of materials and must lie in the range $0 \leq e \leq 1$. Note that if $e = 1$, this reduces to elastic collision as given in (7) and (8). If $e = 0$, then momentum is still conserved, but the loss of kinetic energy is maximized. In this case, the resulting speeds of the two objects are identical. They collide and then travel together, not bouncing at all.

Note that if $m_1 = m_2$, then these equations reduce to

$$v_1' = (v_1(1-e) + v_2(1+e))/2 \tag{11}$$

$$v_2' = (v_2(1-e) + v_1(1+e))/2. \tag{12}$$

Another useful special case is where one of the masses is fixed (it cannot be moved), so the other will bounce off it. This follows from (9) and (10) if we let $v_2 = 0$ and determine the limit as $m_2 \to \infty$. In this case, we find

$$v_1' = -ev_1\ . \tag{13}$$

Mass 1 simply reverses direction upon collision and loses speed by factor $e$. This makes it clear why we restrict $e$ to $0 \leq e \leq 1$.

### 5.1 Dirac Delta Functions

Stewart [54] points out that many of the difficulties in modeling collisions are a consequence of overly restricting the mathematical domains that are used. Impulsive forces, for example, can be naturally modeled using the **Dirac delta function**, a function $\delta : \mathbb{R} \to \mathbb{R}^+$ given by

$$\forall\, t \in \mathbb{R},\ t \neq 0, \quad \delta(t) = 0, \quad \text{and}$$

$$\int_{-\infty}^{\infty} \delta(\tau)\, d\tau = 1.$$

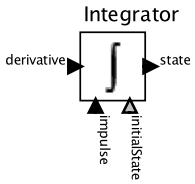


**Fig. 5.** The Integrator actor in Ptolemy II.

That is, the signal value is zero everywhere except at $t = 0$, but its integral is unity. At $t = 0$, therefore, its value cannot be finite. Any finite value would yield an integral of zero. This is indicated by $\mathbb{R}^+$ in the form of the function, $\delta : \mathbb{R} \to \mathbb{R}^+$, where $\mathbb{R}^+$ represents the **extended reals**, which includes infinity. Dirac delta functions are widely used in modeling continuous-time systems (see [30], for example), so it is important to be able to include them in simulations.

Suppose that a signal $x$ has a Dirac delta function occurring at time $t_1$ as follows,

$$x(t) = x_1(t) + K\delta(t - t_1),$$

where $x_1$ is an ordinary continuous-time signal, and $K$ is a scaling constant. Then

$$\int_{-\infty}^{t} x(\tau)d\tau = \begin{cases} \int_{-\infty}^{t} x_1(\tau)d\tau & t < t_1 \\ K + \int_{-\infty}^{t} x_1(\tau)d\tau & t \geq t_1 \end{cases}$$

The component $K\delta(t - t_1)$ is a Dirac delta function at time $t_1$ with weight $K$, and it causes an instantaneous increment in the integral by $K$ at time $t = t_1$.

The Ptolemy II Integrator actor, shown in Figure 5, directly supports Dirac delta functions. Specifically, the actor accepts a discrete-event signal at the input port labeled "impulse," and it interprets the real time of each event that arrives at that port as the time offset of the Dirac delta function ($t_1$ above) and the value of the event as the weight of the Dirac delta function ($K$ above).

Why not include the Dirac delta function on the ordinary input port of the Integrator actor, the one labeled "derivative" in Figure 5? There are two reasons for providing a distinct input port for Dirac delta inputs vs. ordinary continuous-time inputs. First, the value of a Dirac impulse at the time it occurs is not a real number, so we would need some extended data type to include such weighted non-real values. But more importantly, at a superdense time $(t, m)$, the output of the Integrator does not depend on the value of the input at the "derivative" input port, but it *does* depend on the value of the input at the "impulse" port! There is **direct feedthrough** from the impulse input to the output. The Integrator actor is both strict and nonstrict, depending on which input is being considered.

This causality distinction is essential to the soundness of our modeling approach. In fact, we will show below that many problematic modeling problems with discrete physical phenomena manifest as causality loops, and failing to make this distinction obscures this defect. For example, any direct feedback from the output of the Integrator

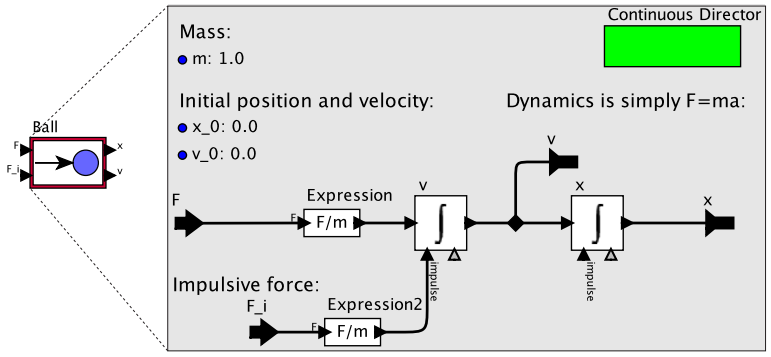


**Fig. 6.** A Ptolemy II model of a ball.

to the impulse input will result in a causality loop, and hence a nonconstructive model. Glockner [19] also advocates separately treating ordinary continuous-time signals and impulsive signals, whose models "split into the atomic and the Lebesgue part."

### 5.2 Modeling Collisions as Impulses

Figure 6 shows a Ptolemy II composite actor that models Newton's equations of motion, $F = ma$. The model has three parameter, the mass $m$ of the ball, and the initial position $x_0$ and velocity $v_0$. The model uses Newton's second law to output the velocity $v$ and position $x$ as a function of time. There are two inputs, a real-valued force $F$ and an impulsive force $F_i$. $F_i$ is required to be a piecewise-continuous discrete-event signal, and its value represents the weight of a Dirac delta function.

A model that composes two instances of the ball model from Figure 6 is shown in Figure 7. The figure also shows a plot of the positions of two balls of diameter 1.0, where the left ball has an initial velocity of 1.0, and the right ball is standing still. After the collision, the situation is reversed. At the superdense time of the collision, the LevelCrossingDetector actor outputs an event, which enables execution of the **CalculateImpulsiveForce** composite actor. That actor is an instance of a subclass of **EnabledComposite**, which executes the inside model only when the *enable* port at the bottom has a present input with value true. The CalculateImpulsiveForce actor samples the current velocities of the balls and calculates the impulsive force that will change the velocities to those given by equations (9) and (10). The impulsive forces are then routed through a pair of **MicrostepDelay** actors, which apply the forces in the next microstep. Without these MicrostepDelay actors, we would have a causality loop, because the CalculateImpulsiveForce actor observes the velocities of the balls, and an impulsive force directly affects the velocities.

A collision occurs when the position of the right edge of the left ball coincides with the left edge of the right ball, and when the velocity of the left ball is greater than the velocity of the right ball. If ball 1 is on the left and ball 2 on the right, and the diameter

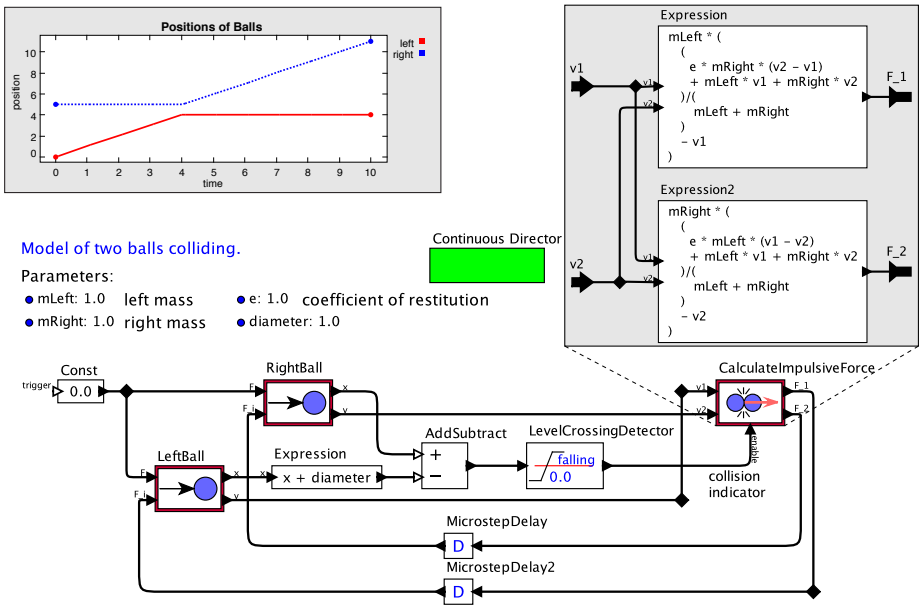


**Fig. 7.** A Ptolemy II model of two balls that collide. [online]

of the left ball is $d$, then the collision occurs when

$$(x_1 + d \geq x_2) \wedge (v_1 > v_2).$$

However, this statement is fundamentally problematic. A collision occurs at the instant when the above predicate becomes true. First, there may be no such precise instant (suppose the balls are initially touching and we start applying a force to the left ball). Second, computational numerical methods have to approximate the continuums of time and position. In practice, to model such a collision as a discrete event, we need an error tolerance. Lower error tolerance will translate directly into increased computational cost.

In Figure 7, the collision is detected by an actor labeled LevelCrossingDetector, which detects zero crossings of the distance between the two balls. This actor collaborates with the solver to adjust the step size of the numerical ODE solution so that the zero crossing is pinpointed with precision specified by a parameter.

Detecting collisions as zero crossings of the distance function, however, raises another difficulty. Specifically, if two balls are initially touching, the distance starts at zero. If a collision occurs, it does not cross zero. We consider this problem next.

### 5.3 Simultaneous Collisions

Consider the scenario shown in Figure 8, which is analogous to Newton's cradle. Two balls are initially touching, with zero distance between them, and a third ball approaches

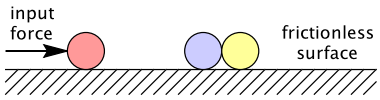


**Fig. 8.** Three balls on a frictionless surface.

them from the left. At the instant of the collision, the left ball will transfer its momentum to the middle ball (assuming it has the same mass), which will then instantly transfer its momentum to the right ball. These two transfers occur at successive superdense time microsteps, so that the total momentum in the system is constant over time.

The zero-crossing detection strategy in Figure 7, however, will not work for this scenario. It will fail to detect the second collision, because the distance between the middle and the right balls does not cross zero. It is initially zero, and a model like that in Figure 7 will show the left ball passing through the right ball, as shown in Figure 9(a). This difficulty is corrected by using a more sophisticated collision detection shown in Figure 10, which yields the plot in Figure 9(b). This correctly emulates Newton's cradle. This collision detector declares a collision to occur whenever the distance between the

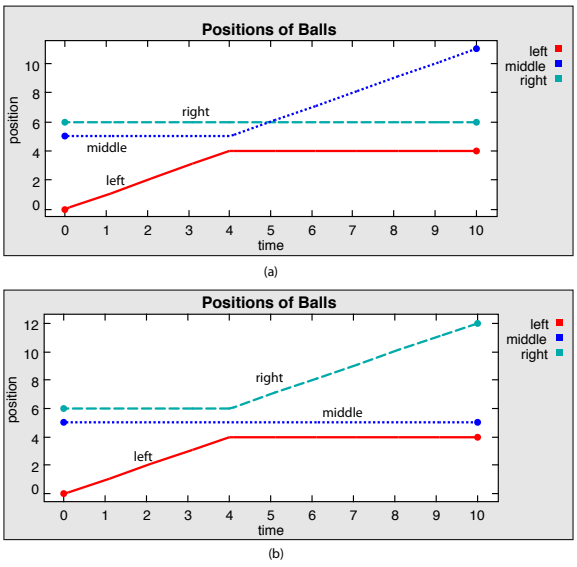


**Fig. 9.** (a) Second collision is not detected. [online] (b) Second collision is detected. [online]

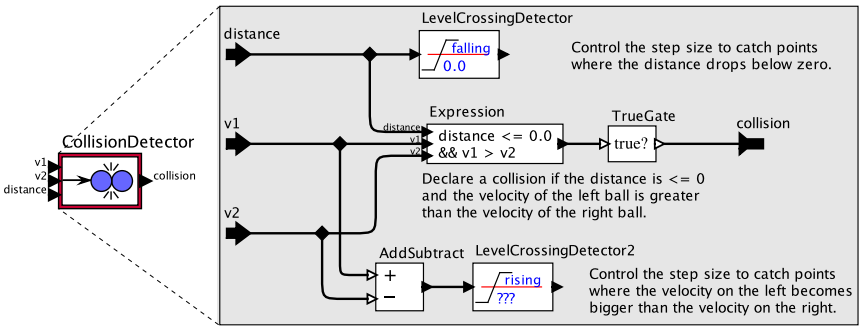


**Fig. 10.** A Ptolemy II model of a collision detector.

balls is less than or equal to zero and the velocity of the left ball is greater than the velocity of the right ball.[3]

Consider a third scenario, where two balls simultaneously collide with a stationary ball from opposite sides, as shown in Figure 11. This scenario is fundamentally problematic, and the Newtonian model of collisions given above has difficulty with it. A naive model superimposes the impulsive forces from the two simultaneous collisions, which cancel each other out in the middle ball. The result is a model where upon colliding, all three balls instantly stop, as shown in Figure 12(a). All the energy in the system is instantly lost!

One possible solution is to replace Newton's model with the **Poisson hypothesis**, which postulates that a collision consists of two distinct phases, a **compression phase** and a **restitution phase**. It is possible to construct a model where the two collisions have simultaneous compression phases, storing their kinetic energy as potential energy, and then, one superdense time index later, simultaneously release the potential energy as kinetic energy. Such a model would seem to solve the problem, but actually, it doesn't. There are many ways to assign kinetic energy such that both energy and momentum are conserved. In fact, such a solution simply masks a more fundamental physical problem.

[3] Note that the signal out of the Expression actor in Figure 10 is not, in fact, piecewise continuous, so we must use caution and avoid presenting this signal or any signal triggered by it directly to a numerical ODE solver. MicrostepDelay actors used as in Figure 7 will convert a discrete signal that is not piecewise continuous into one that is (its output at microstep zero will always be absent).

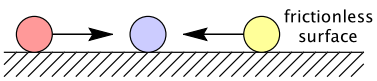


**Fig. 11.** Three balls with simultaneous collisions.

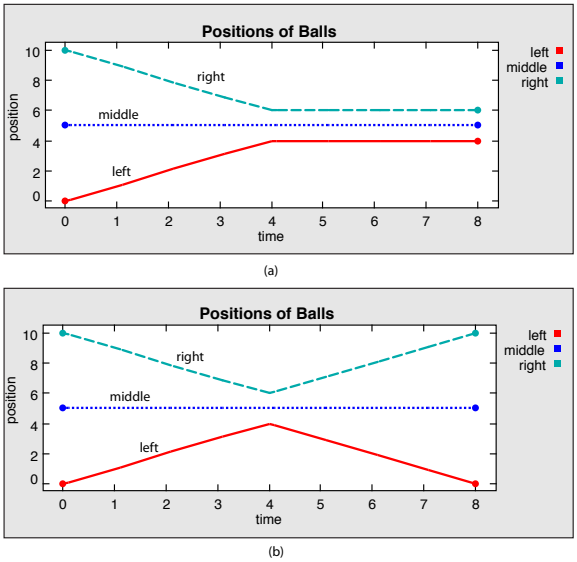


**Fig. 12.** (a) Superposition of simultaneous impulsive forces results in instant loss of energy. [online] (b) Interleaving of simultaneous collisions at distinct microsteps conserves energy. [online]

An alternative solution is consider the two simultaneous collisions as being arbitrarily interleaved. That is, one occurs first, then the other. If the balls have the same mass, then it does not matter which one occurs first, and the model yields reasonable behavior. The outside balls bounce back with equal speed. This behavior is shown in Figure 12(b). The model that generates this plot arbitrarily chooses one of the collisions, ignoring the other, calculates the modified velocities, and then, one superdense index later, detects the second collision, which occurs with the modified velocities.

The portion of the model that arbitrarily selects a collision is shown in Figure 13. This model uses the **Default** actor, which behaves as follows. If an input is present on its *preferred* input port, then that input is passed through to the output. Otherwise, the *alternate* input, if present, is passed through to the output. The model also uses the **Inhibit** actor, which passes the left input to the output unless the bottom input port is present. This actor ensures that the second collision is completely ignored if the first occurs at the same superdense time index. In the next superdense time index, after the first collision has resulted in new velocities, a new collision may occur.[4]

However, if the balls have different masses, then the behavior depends on the order in which the collisions are handled, even though no time elapses between collisions.

[4] Note that this model arbitrarily chooses one of the collisions, but it always prefers one over the other. This is acceptable for a nondeterministic model, but may not be acceptable in simulation. We may instead want to choose one of the two collisions probabilistically. We leave this as an exercise.

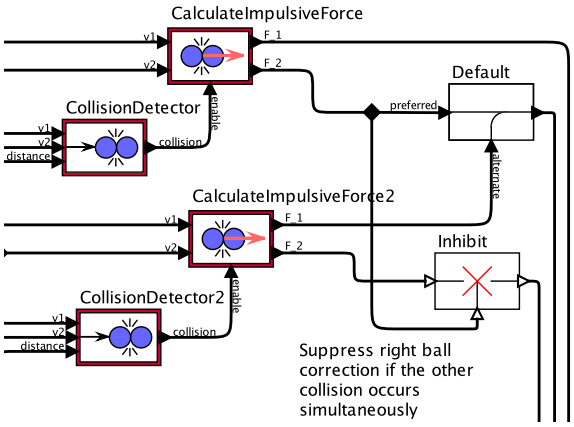


**Fig. 13.** A portion of the model producing the plot in Figure 12, showing the arbitrary selection of the upper collision when two occur simultaneously. [online]

This is shown in Figure 14. It is easy to verify that both behaviors conserve both momentum and energy, even though they result in different trajectories for the balls. It seems that a reasonable modeling choice would be to nondeterministically choose an ordering.

In light of the Heisenberg uncertainty principle, these difficulties should not be surprising. The Heisenberg uncertainty principle states that we cannot simultaneously know the position and momentum of an object to arbitrary precision. But the reaction to these collisions depends on knowing position and momentum precisely. A direct expression of such simultaneous collisions results in a nonconstructive model. To get a constructive model, we have to insert microstep delays and tolerate nondeterminism. Nature, it seems, resolves nonconstructiveness with uncertainty. Chatterjee and Ruina suggest that indeed, a reasonable and practical approach to simulating such systems is to nondeterministically choose an ordering [14].

Note that doing more detailed modeling of the collisions does not solve the problem. It just shifts the uncertainty to other parts of the model. Unlike the two-ball collision, there are multiple solutions that conserve energy and momentum. We conjecture that defensible detailed models could yield the same (or more) variabilities in behaviors.

It might seem odd to invoke quantum mechanics when considering macro phenomena such as collisions of balls. But the impulsive model we are using has infinite precision, and in physics, it is at high precisions where quantum mechanical effects become important. Moreover, the conjecture here that nonconstructive models associate with quantum mechanical uncertainty is perhaps related to the study of nonconstructive digital circuits in [36,53] and the proof in [39] that such models correspond to physical circuits that are vulnerable to unstable oscillations.

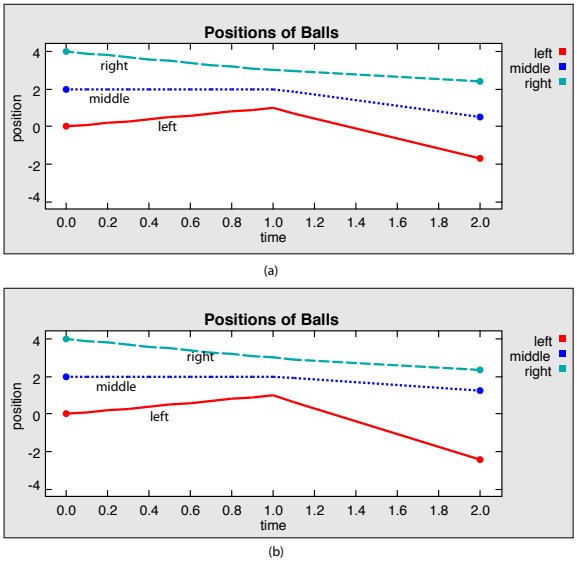


**Fig. 14.** (a) Interleaving of simultaneous impulsive forces where the collision of the left ball (with mass 0.2kg) with the middle ball (with mass 1kg) is handled before the collision of the right ball (with mass 5kg) with the middle ball. [online] (b) Interleaving the collisions in the opposite order yields different behavior. [online]

Notice that the three-ball collisions of Figure 9(b) do not have the same difficulty because of the linear ordering of causal relationships. Nevertheless, even the notion of causality is fraught with difficulty. For a wonderful philosophical discussion of this issue, see [50], which includes an essay arguing that there is no basis in physics for the notion of causality [44]. Causality, it claims, is a human cognitive construction.

### 5.4 Collisions vs. Pushing

Consider again just two balls on a frictionless surface. Suppose the two balls are initially stationary and touching, so $x_1 + d = x_2$. Suppose now that we begin to apply a force of one Newton at time $t = 1$,

$$F(t) = \begin{cases} 0, t < 1 \\ 1, t \geq 1 \end{cases}$$

The left ball begins moving to the right at time 1. Then when does the collision occur? At time 1, the velocity of both balls is zero, so the collision does not occur at time 1. The collision occurs at the smallest time (a real number) greater than 1. There is no such time, of course, so again, we have to choose a numerical precision with which to approximate this behavior.

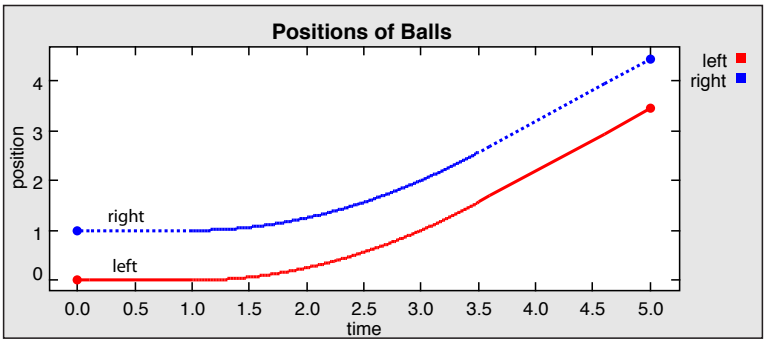


**Fig. 15.** Positions of two balls that start out touching and not moving. From time 1, a constant force is applied to the left ball, which results in pushing both balls to the right. At time 3.5, we stop applying the force, so the balls drift together at the same velocity. [online]

This scenario can be modeled by the model in Figure 7, but with the LevelCrossingDetector replaced with the CollisionDetector of Figure 10. Figure 15 shows the resulting plot of the positions of the balls over time, and Figure 16 shows the velocities. Note from the detail of the velocities that in this model, the left ball repeatedly collides with the right ball, providing an impulsive force each time. So the velocity of the right ball is piecewise constant. The time interval between collisions is determined by the precision with which zero-crossing detection is being done, which in this case is $10^{-4}$ seconds.

The computational cost of such a simulation is substantial. For high precision, the density of collisions must be high. This will force a simulator to take small step sizes. A better model is a modal model, where the balls are modeled as a single mass when the left ball is accelerating to the right and they are touching, and as two distinct balls when they are separated. We provide such a model in Section 6 below.

Erleben et al. [17, p. 154] give an interesting example of two masses on a rigid immovable surface, one on top of the other. They assume perfectly inelastic collisions, where the coefficient of restitution is $e = 0$. So when the masses collide, they move together at a velocity the conserves momentum. Suppose you apply an impulsive downward force on the top mass. The top mass will instantly acquire a downward velocity. It immediately collides with the lower mass, so the two masses acquire the same downward velocity. The lower mass then collides with the rigid surface, which is modeled as an infinite mass, so the lower mass stops moving. The upper mass then immediately collides with the lower mass, the two masses again acquire a downward velocity. The cycle repeats infinitely without time advancing and without the momentum ever becoming identically zero. This is a chattering Zeno model. We now explore more classical Zeno models.

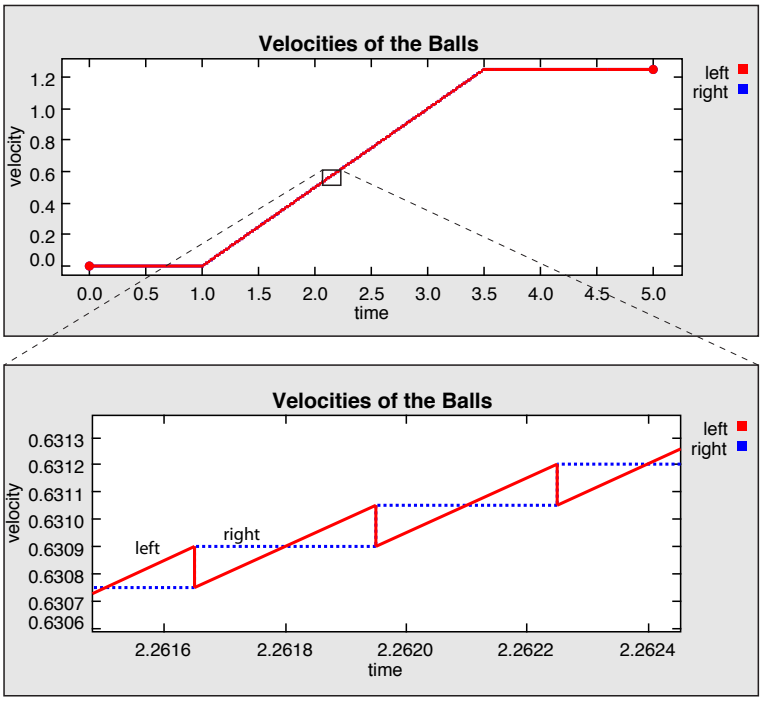


**Fig. 16.** Velocities of the two balls in Figure 15, showing that the velocity of the right ball is piecewise constant, and that it is getting pushed by repeated discrete collisions.

### 5.5 Zeno Conditions

Modeling instantaneous events inevitably brings up the question of Zeno conditions, named after Zeno of Elea, a pre-Socratic Greek philosopher famous for his paradoxes. A model that exhibits Zeno behavior has an infinite number of events in a finite time. Any execution of the model that constructs all these events in temporal order will fail to advance time beyond a certain point.

First, we would like to point out that Zeno conditions are not a consequence of discrete events. Continuous-time systems can also exhibit Zeno behaviors. Consider for example the function $x\colon \mathbb{R} \to \mathbb{R}$ defined by

$$x(t) = \begin{cases} \sin(2\pi t/(1-t)) & 0 \le t < 1 \\ 0 & \text{otherwise} \end{cases}$$

A plot of this function over the time range $[0,2]$ is shown in Figure 17. There are an infinite number of oscillations prior to time 1.0.

Nevertheless, models with discrete events seem to be more vulnerable to Zeno conditions. We have already mentioned the possibility of chattering Zeno behavior, where

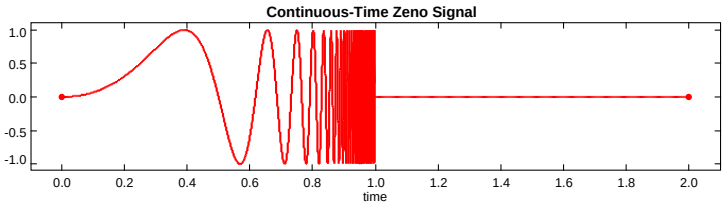


**Fig. 17.** Continuous-time signal with Zeno behavior.

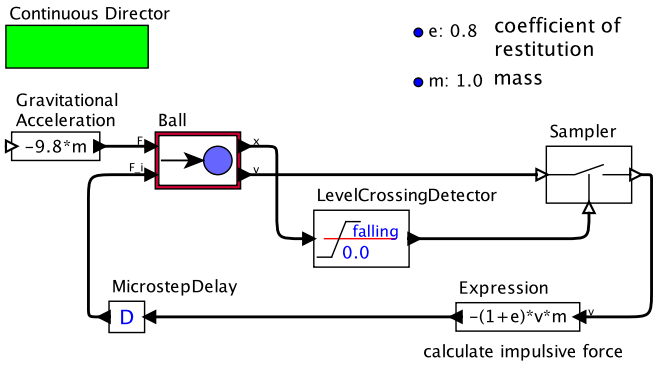


**Fig. 18.** Bouncing ball using impulsive forces. [online]

a signal fails to converge to a final value at a particular model time. A classic example of a Zeno system that is not a chattering-Zeno system is an idealized bouncing ball. In such an idealized model, a ball collides with a fixed surface and bounces according to equation (13). That is, upon collision, the velocity reverses instantaneously and is attenuated by a coefficient of restitution.

An implementation of such a model is shown in Figure 18. A constant gravitational acceleration of 9.8 m/sec$^2$ is applied to an instance of the same ball model in Figure 6. The model in Figure 18 uses a LevelCrossingDetector actor to determine when the ball collides with the surface, and then calculates the weight of an impulsive force that will reverse the velocity according to (13). As before, we need a MicrostepDelay in the feedback loop. This specifies that the impulsive force should be applied in the next microstep after the detection of the collision. Were it not there, we would have a causality loop, where the velocity output of the Ball model depends directly on the impulsive force.

A plot of the positions and velocities produced by the model are shown in Figures 19 and 20. These plots exhibit the discontinuous changes in velocity at the times of collision. But note an interesting phenomenon: the ball drops below the surface around time 12.8. The ball appears to “tunnel” through the surface.

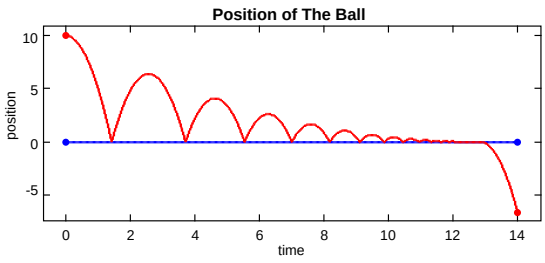


**Fig. 19.** Plot of the position of the bouncing ball vs. time.

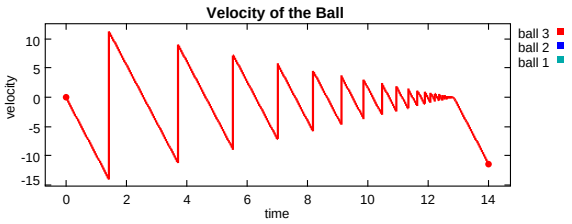


**Fig. 20.** Plot of the velocity of the bouncing ball vs. time.

In fact, this model has Zeno behavior. Analytically, we can determine that the number of bounces before time 13 is infinite. So why does our simulation show the ball tunneling through the surface and going into a free fall?

The LevelCrossingDetector actor has an errorTolerance parameter. Such a tolerance is intrinsic to any numerical technique for detecting level crossing of a signal in a continuum. This tolerance allows the simulation of the bouncing ball's position to numerically drop below zero by a small amount. When its velocity is low enough, then when it bounces, it will not have enough energy to rise again above zero, and hence no further zero crossings will occur. The ball goes into a free fall. Figure 21 zooms in on the point in time where this occurs. You can see in the figure that the final bounce fails to rise above the surface.

The errorTolerance of the LevelCrossingDetector in this plot is set to $10^{-4}$. We can adjust the errorTolerance and constrain the step size of the simulator arbitrarily, up to numerical precisions of double-precision floating point numbers. However, no choice of precision constraints will prevent this tunneling.

Again, we can invoke the Heisenberg uncertainty principle. On physical grounds alone, this model does not make sense at arbitrary precisions. The execution gets to a state of the system where the model is no longer valid. To prevent this, we need to use modal models, as described below.

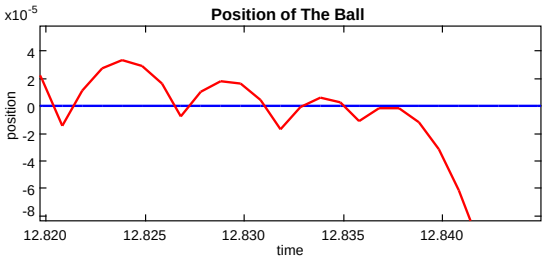


**Fig. 21.** Detail of the point in simulation where the bouncing ball tunnels through the surface.

One possible interpretation is that the model is flawed in that the surface on which the ball is bouncing is infinitely thin. Indeed, if a physical system had this property, then quantum mechanical tunneling suggests that indeed the ball could drop through the surface. Perhaps tunneling is nature's way of preventing Zeno conditions from stopping time! Less speculatively, we need to switch to a more suitable model when the velocity of the ball at the time of collision drops below a threshold related to the accuracy of collision detection.

## 6 Modal Models

No (useful) model has high fidelity over all possible operating conditions and all possible configurations of the system. Modal models provide a mechanism for having more than one model of a physical system, and for switching between models when the operating conditions change or when the configuration of the system changes. Modal models provide an operational semantics for hybrid systems [31] that is compatible with superdense time. In this section, we briefly describe the structure of modal models, and then give a sequence of examples that resolve several issues raised above.

### 6.1 The Structure of Modal Models

The general structure of a modal model is shown in Figure 22. The behavior of a modal model is governed by a state machine, where each state is associated with a **mode**. In Figure 22, each mode is represented by a bubble (like a state in a state machine) but it is colored to indicate that it is a mode rather than an ordinary state. A mode, unlike an ordinary state, has a **mode refinement**, which is a submodel that defines the mode's behavior.

Like states in a finite state machine, modes are connected by arcs representing transitions with guards that specify when the transition should be taken. A **guard** is a boolean-valued expression that may reference inputs to the modal model (in1 and in2 in the figure) and/or outputs of the refinements (out in the figure). Guards may be

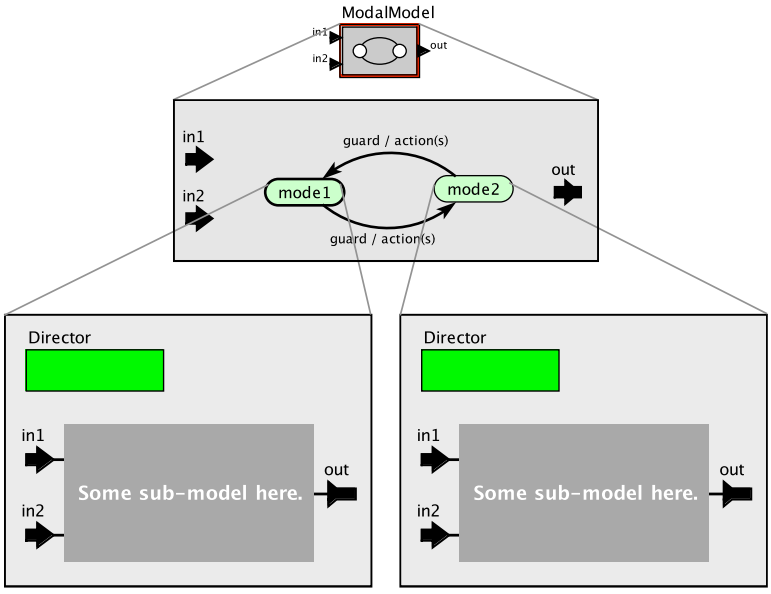


**Fig. 22.** General pattern of a modal model with two modes.

interpreted as "enabling" a transition or as "triggering" a transition. In the former interpretation, a transition *may* be taken when the guard evaluates to true. In the latter interpretation, a transition *must* be taken when the guard evaluates to true. We adopt the latter interpretation here in order to be able to specify deterministic models.

The transitions can also have one or more **action**s. In this paper, we will use actions to initialize the state of the destination mode. This will provide the initial conditions for execution in that mode.

Many variants of modal models have appeared in the literature and in simulation tools. In this paper, we use the specific modal models of Ptolemy II, which are described in detail in [18]. The semantics of these models is given in [29].

### 6.2 Bouncing Ball

The bouncing ball example in Figure 18 exhibits tunneling through the bounce surface, as shown in Figure 19. The problem is that the impulsive model of collisions is invalid below a velocity and position threshold. A modal model solution to this problem is shown in Figure 23, with resulting plot in Figure 24. This model has two modes of operation, called "falling" and "sitting." The falling mode is identical to that in Figure 18, but the sitting mode provides a much simpler model, one that is suitable for a ball that is just sitting on a surface. The transition from falling to sitting is triggered when the absolute value of position and velocity are both below a parametric threshold $p$. Upon taking the transition, the position $x$ of the destination mode is set to the current position,

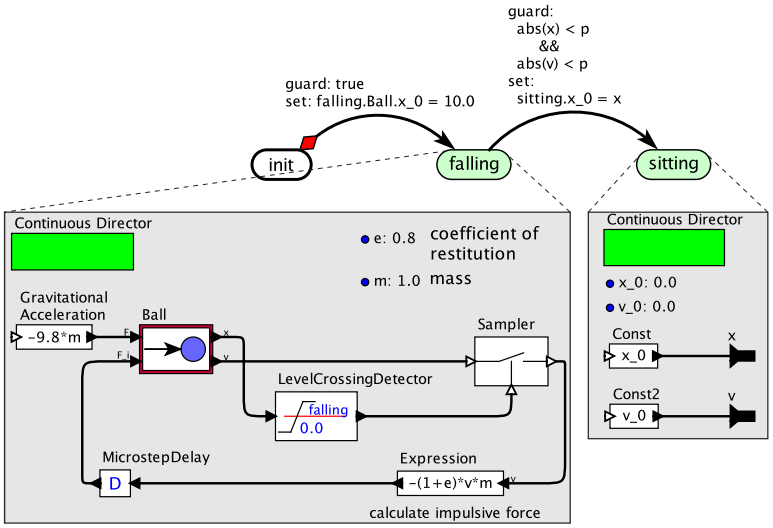


**Fig. 23.** Modal version of the bouncing ball example that avoids tunneling. [online]

which will typically not be exactly zero. This transfer of state prevents a discontinuity in position, albeit a small one.

The ability to set the initial conditions of the destination mode is quite useful. In fact, we can use this to get the same effect as impulsive forces, as shown in Figure 25. In that figure, the bounce itself is handled by a transition instead of a Dirac delta. Instead of going to a new mode, the transition goes back to the same "falling" mode. The transition adjusts the velocity of the ball according to (13), but otherwise leaves the state of the refinement unchanged (the position of the ball, for instance, does not change when this transition is taken). The transition uses a notation inspired by SyncCharts [3], where the circled "H" indicates that this is a **history transition**. When a history transition is taken, the destination mode is not initialized; its state is the same after the transition as before, except for any specific actions that are specified on the transition. Notice that this model no longer requires any microstep delays.

### 6.3 Collisions vs. Pushing

Recall from Section 5.4 the scenario where two stationary balls are initially touching, and an external force is applied to the left ball starting at time 1. The resulting velocities of the balls are shown in Figure 16, which shows that the impulsive model results in many small collisions, yielding a rather inefficient simulation.

Figure 26 shows a modal version of this model, where initially the system is in the *together* mode, which models the balls as shown in the bottom of the figure as a single mass that travels together. If at any time the force on the right ball is greater

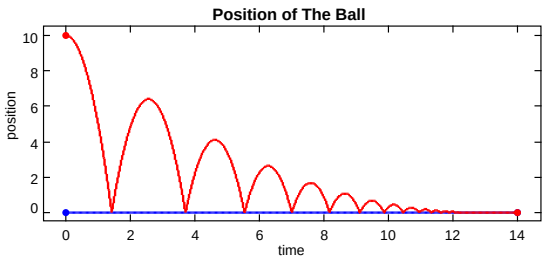


**Fig. 24.** Position of the bouncing ball example that avoids tunneling.

than the force on the left ball or either ball receives an impulsive force, then the system transitions to different mode where the balls are separate. Figure 26 doesn't show the details, but as part of this transition, the state of the two balls should be initialized. The result of this model is a plot with the same shape as the top part of Figure 16, but without many small collisions. The simulation is far more efficient, since the step sizes are now determined only by the integration accuracy, and not by the time between collisions.

An interesting subtlety about this model concerns the transition back from the *separate* mode to the *together* mode. In Figure 26, the guard for this transition requires the balls to be touching and their velocities to match. The test checks for equality, and equality is rarely a good test for floating point numbers that are approximating real

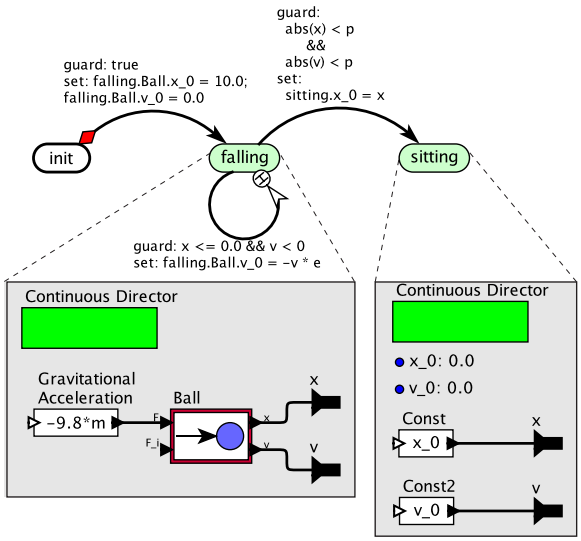


**Fig. 25.** Alternative modal version of the bouncing ball. [online]

numbers. Indeed, in a model like this, the transition back to the *together* mode would be rare indeed. For elastic collisions, this conforms with intuition. It is very difficult to prevent two elastic objects from bouncing off one another. A more interesting scenario, however, would model some "sticking" between the balls. Such a model is a part of the standard Ptolemy II demo suite, is described in detail in [28, section 4.2.2], and is available online.

It is tempting to similarly use mode transitions to handle simultaneous multibody collisions, like the scenario studied in Figures 11, 12(b), 13, and 14, where two balls simultaneously collide with a stationary ball from opposite sides. It is difficult to make such a model modular, however.

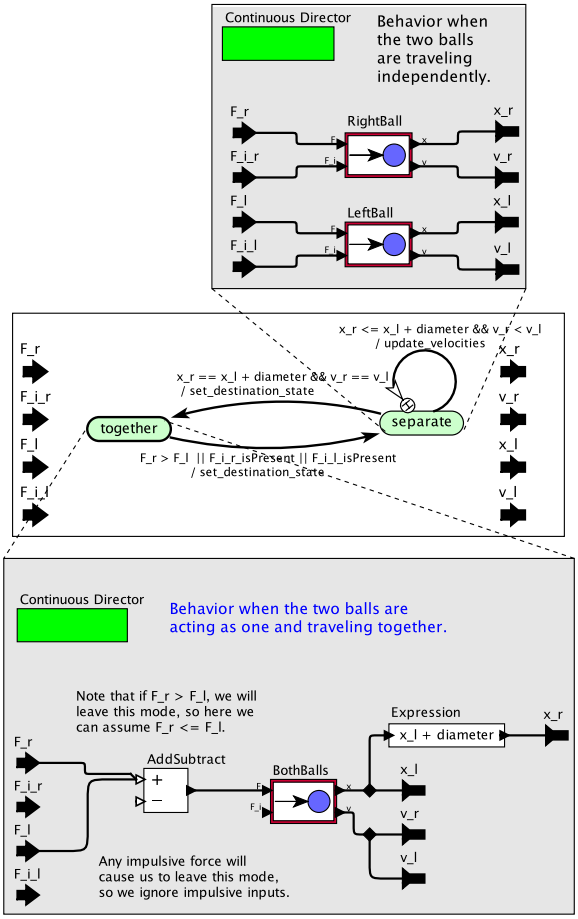


**Fig. 26.** Modal version of the model that produces the plot in Figure 16. [online]

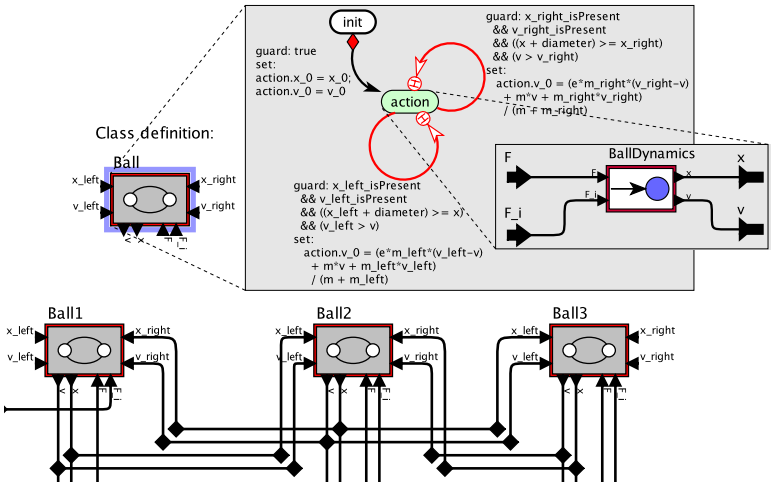


**Fig. 27.** An attempt at a modal model for simultaneous collisions like those in Figure 11. [online]

An attempt is shown in Figure 27, but this attempt is flawed. In this model, three balls communicate their positions and velocities to each other. There is no causality loop induced by this communication because the positions and velocities are outputs of integrators. When a collision occurs, a transition is taken from the *action* state back to itself. If two collisions occur simultaneously on opposite sides, then one of the transitions is chosen nondeterministically. However, in this model, each ball independently detects collisions and adjusts its own velocity. Hence, when the center ball ignores one of two simultaneous collisions, the ball that it is ignoring will not ignore the collision. That ball adjusts its velocity without a corresponding adjustment occurring in the center ball.

One solution might be for the center ball to communicate to its neighbors which of two simultaneous collisions it chooses. However, this solution will not scale. If we have four balls instead of three, then such communication will form a causality loop, and the model will be non-constructive. Another solution is to arbitrarily choose left collisions over right collisions (see online for such a solution). Such a solution works well in one dimension, but in two or three dimensions, it will be more difficult. In one dimension, we have at most two simultaneous collisions with each ball, but in more dimensions, we could have more collisions.

Any general interleaving solution will have to handle all simultaneous collisions together, using centralized logic to choose the interleaving, and using microstep delays to communicate the choices. The real issue here is that we are trying to build a model that violates fundamental laws of physics by trying to define simultaneous impulsive collisions. It should not be surprising that constructing such models is difficult. Note that it will not help to define the model in an acausal language like Modelica. In fact,

that will just obscure the problem. A causal language has the advantage that it is easier to see why a model is non-constructive.

### 6.4 Friction

Friction is a sufficiently complex physical phenomenon that calculation from first principles is impractical. Instead, engineers use empirical methods that are based on experiment. A commonly used empirical model of friction dates back to Leonardo da Vinci in the 15th century, and was further developed by Guillaume Amontons and Charles-Augustin de Coulomb in the 17th and 18th centuries. We will refer to it here as the **Coulomb model of friction**.

The Coulomb model distinguishes **static friction** from **kinetic friction**. In the former, the two objects move together (their relative velocity is zero), and the force of friction is just enough to keep them moving together. In the latter, the two objects have non-zero relative velocity, so they are sliding against one another. Kinetic friction is a force opposing their relative motion. The kinetic friction force has three basic properties:

1. it is directly proportional to the force pressing the two objects together;
2. it is independent of the apparent area of contact; and
3. it is independent of the sliding velocity.

For the first property, the proportionality constant is the kinetic **coefficient of friction** $\mu_k$, an empirical quantity found through measurement.

Consider two objects with masses $m_1$ and $m_2$ arranged as shown in Figure 28. Here, mass 1 slides on a frictionless surface in response to an input force. Mass 2 will slide with mass 1 if the input force is sufficiently small. Specifically, if mass 1 moves, Coulomb's model states that static friction will apply a force $F_2$ on mass 2 in the direction of motion that satisfies the following inequality,

$$F_2 \leq \mu_s F_n$$

where $F_n$ is the **normal force** pressing the two objects together, and $\mu_s$ is the static coefficient of friction. Specifically, this force will be just enough to get mass 2 to match the acceleration of mass 1, causing the two masses to behave as one. If the input force on mass 1 is $F_1$, then Newton's second law tells us that the acceleration of the two masses together will be $F_1/(m_1+m_2)$. To keep the masses together, this requires a force on

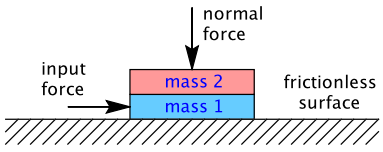


**Fig. 28.** Two masses with friction.

mass 2 due to static friction of

$$F_2 = \frac{F_1 m_2}{m_1 + m_2}.$$

Sliding will begin when $F_2$ exceeds $\mu_s F_n$. Hence, the critical **breakaway force** on mass 1 is

$$F_b = \frac{\mu_s F_n (m_1 + m_2)}{m_2}. \tag{14}$$

That is, when $F_1 > F_b$, mass 2 will begin sliding. Note that with the arrangement in the figure, $F_n$ is due to gravity, and hence $F_n = m_2 g$, where $g = 9.81\text{m/sec}^2$ is the acceleration of gravity at the earth's surface. Hence, the breakaway force can be written

$$F_b = \mu_s g (m_1 + m_2). \tag{15}$$

Modal models provide a natural way to model friction, where a breakaway is a mode transition. Two masses may also come together again, at a time where their velocities match. This **capture** can be modeled as a mode transition. But such models get particularly interesting when we combine them with impulsive collisions. We do that next.

### 6.5 Combining Collisions and Friction

Stronge [55] gives an excellent analysis and critique of Newton's model of collisions when combined with Coulomb's model of friction, showing that in certain circumstances, the models appear to be incompatible (they create energy). His solution is a much more detailed model of the physics. Our solution is instead to stick to the naive and simple models, but to explicitly limit the use of these simple models to their regime of applicability. We use modal models to switch between models when crossing from one regime to another.

A model that combines friction with impulsive collisions for the scenario of Figure 28 is shown in Figure 29. In this model, we assume that *any* non-zero impulsive force on mass 1 causes a breakaway. This makes physical sense because an impulsive force is instantaneously infinite, and therefore exceeds the breakaway force (14) at the instant of the collision. A weak impulsive force, of course, will result quick recapture. Note that Mosterman et al. [42] take a different approach, where they instead calculate the velocity that would result from breakaway, and if that velocity exceeds a threshold, then they declare the breakaway valid. Otherwise, they reverse the breakaway and treat the two masses as one. This latter approach does not appear consistent with Coulomb's model, because the breakaway threshold is a force, not a velocity. Moreover, it suffers an unnecessary causality loop, where the actual post collision velocity depends on whether the breakaway occurs, and whether the breakaway occurs depends on the actual post collision velocity. This causality loop provides further evidence that their model is based on questionable physics.

The plot in Figure 29 shows a rather complex behavior. A sinusoidal non-impulsive force is applied to mass 1. This force is sufficient to cause a breakaway around time 0.2. After breakaway, a constant force is applied to mass 2 (sliding friction), which causes its velocity to drop linearly. Around time 0.8, mass 1 reverses direction, reversing

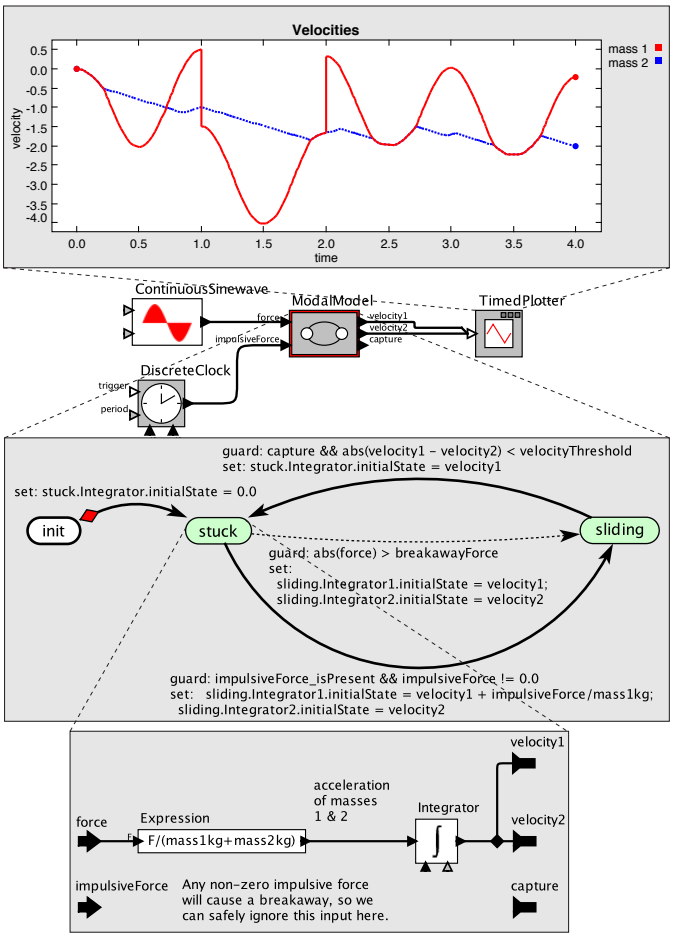


**Fig. 29.** Model of friction with impulsive collisions. [online]

the polarity of the force on mass 2. At time 1.0, an impulsive force produced by the DiscreteClock actor causes an abrupt reversal of the direction of motion of mass 1. Around time 1.8, recapture occurs, followed by another impulsive force at time 2.0, and another recapture near time 2.4.

A breakaway is represented in Figure 29 by the bottom transition from the *stuck* mode to the *sliding* mode. The other transition from *stuck* to *sliding* has a dashed line, which in Ptolemy II means that this transition will be considered only if no solid-line transition is enabled. The dashed transition, therefore, will be taken if there is no impulsive force and the non-impulsive force exceeds the breakaway force. Impulsive forces trump non-impulsive forces. Note that it is safe to ignore the non-impulsive force at the

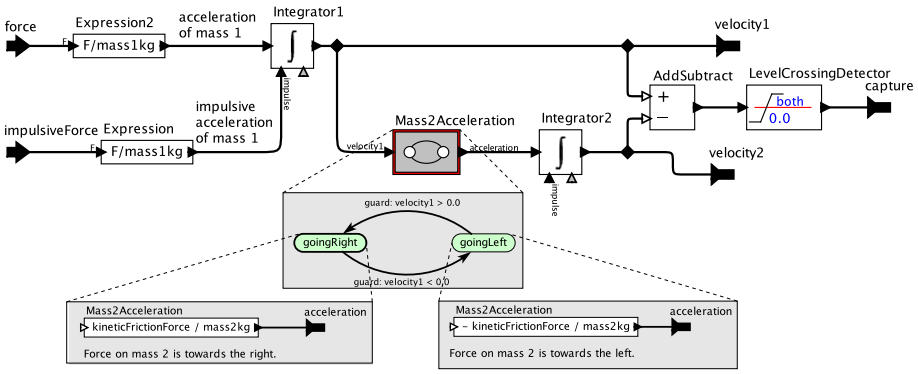


**Fig. 30.** Mode refinement for the sliding mode of Figure 29. [online]

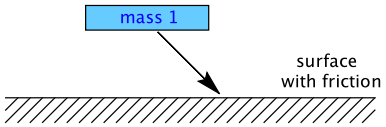


**Fig. 31.** Mass falling at an angle onto a surface with friction.

time that an impulsive force occurs, because no time will elapse before the next tick, and a non-impulsive force has no effect in the microstep dimension of time.

The transition from *sliding* to *stuck* represents capture. It occurs when a *capture* event is detected and the velocity difference is lower than some threshold (a finite precision is required here, because comparing equality of real numbers makes no sense). The capture event is produced by the *sliding* mode refinement, which is shown in Figure 30. That refinement uses a zero-crossing detector to identify when the velocities of the two masses cross in either direction. Why do we need both the velocity difference comparison on the transition and the zero-crossing detector? Because a zero crossing can occur if mass 1 is subjected to an impulsive force that causes its velocity to cross that of mass 2. In the plot in Figure 29, this occurs at time 1.0. This event does not cause recapture. But it does cause the acceleration of mass 2 to abruptly reverse polarity, because the sign of the velocity of mass 1 reverses.

The polarity of the acceleration imposed on mass 2 by dynamic friction is piecewise constant, following Coulomb's principle, but its sign depends on the sign of the velocity of mass 1. Dynamic friction will push mass 2 to the right if mass 1 is moving to the right, and it will push to the left if mass 1 is moving to the left. This reversal of polarity is handled by the simple modal model in Figure 30, where the two mode refinements each produce a constant output. This idiom ensures a piecewise continuous signal.

### 6.6 Relating Impulse Weights

The scenario of Figure 28 has some interesting limiting cases. If the static coefficient of friction is infinite, then the two masses are permanently stuck together. In this case, they act as one mass. Similarly, if the dynamic coefficient of friction is infinite, then mass 2 will be recaptured immediately after any breakaway. But even in these cases, we could assign a weight to the coefficients of friction. An impulse force could therefore "break" a monolithic mass into two pieces if the weight on the impulsive force exceeds the weight on the (infinite) breakaway force.

Erleben et al. [17, p. 154] study an interesting example where a mass falls at an angle onto a rigid immovable surface, as shown in Figure 31. In this case, at the time of the collision, the impulsive force has both a vertical and a horizontal component. The vertical component imposes an instantaneously infinite normal force, so the breakaway force given by (14) is infinite. This suggests that since the surface is immovable (its mass is infinite), the falling mass should instantly stop. But the horizontal component also imposes an instantaneously infinite force, which suggests that the mass should break away and start sliding. In this case, the test for breakaway is comparing two infinite values. Fortunately, by modeling these impulses as Dirac delta functions, we still have a legitimate basis on which to compare them, which is to compare the weights of the two impulses. If the weight of the horizontal impulse exceeds the weight of the breakaway force, then the mass should start sliding upon collision. Otherwise, it should stop. This is consistent with the solution given by [17].

## 7 Mode-Dependent Causality

In the models above, causality is relatively independent of mode. Velocities and positions are always outputs produced in reaction to forces, which are inputs. These facts do not change with the mode changes. However, in more interesting models, different modes of the same system can exhibit different causal relationships between variables. This adds considerable complexity to modeling and simulation. We study this problem with examples from electrical circuits that have elements that are modeled with discrete behaviors. Diodes and switches are two commonly used examples of such elements.

### 7.1 Dual-Mode Diode Circuit

Consider the "oscillating LCD" circuit in Figure 32. This circuit is considered by Benveniste et al. in [8]. The voltages across the three circuit elements, which are a function of time, are $u$, $w$, and $v$ respectively, with the polarities shown in the figure. **Kirchoff's voltage law** tells us that at all times $t$,

$$u(t) + w(t) + v(t) = 0. \tag{16}$$

The current flowing around the cycle is $i$, with the polarity shown in the figure. The capacitor relates the voltage $v$ with the current $i$ according to

$$i(t) = C\frac{dv(t)}{dt}, \tag{17}$$

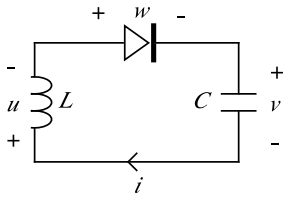


**Fig. 32.** Oscillating LCD circuit from [8].

where $C$ is the capacitance. I.e., current is proportional to the rate of change of voltage. The inductor relates the voltage $u$ and the current $i$ by

$$u(t) = L\frac{di(t)}{dt}, \tag{18}$$

where $L$ is the inductance. I.e., the voltage is proportional to the rate of change of current.

The diode is what makes this circuit interesting. One model for an ideal diode is that it is either **reverse biased**, in which case

$$\begin{aligned} i &= 0 \\ w &< 0, \end{aligned}$$

or it is **forward biased**, in which case[5]

$$\begin{aligned} i &> 0 \\ w &= 0. \end{aligned}$$

The diode, therefore, defines two modes for this circuit. In the forward biased mode, since $w = 0$, (16) implies that

$$u(t) = -v(t),$$

hence

$$\frac{di(t)}{dt} = \frac{-1}{L}v(t),$$

and

$$\frac{dv(t)}{dt} = \frac{1}{C}i(t).$$

These two equations are realized in the model shown in Figure 33 in the forwardBiased mode. The integrator labeled "Capacitor" represents the capacitor, which stores voltage, and the integrator labeled "Inductor" represents the inductor, which stores current.

[5] Note that a more realistic diode model will exhibit a non-zero voltage drop when it is forward biased. This model is easily generalized to accommodate this, as we will do with the next example.

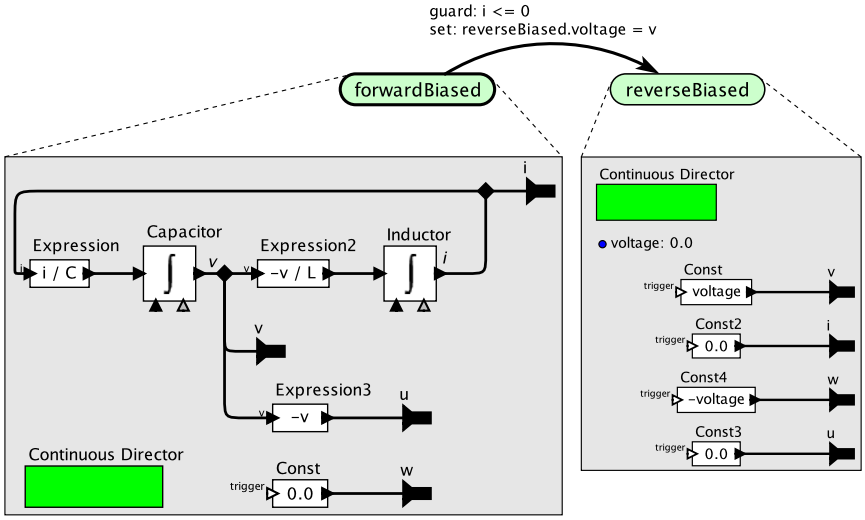


**Fig. 33.** Modal model for the LCD circuit from [8]. [online]

When the current hits zero, the diode instantly becomes reverse biased, and the model makes a transition to the reverseBiased mode. This mode contains a very different model of the circuit. All of the outputs are constant, as shown in the plot in Figure 34.[6]

The constant outputs follow from the equations above. Specifically, since the current becomes a constant zero, the derivative of the current also becomes zero, and hence, from (18), the voltage across the inductor also becomes zero. Since the current becomes zero, (17) implies that the voltage across the capacitor becomes constant. Finally, since $u = 0$, (16) implies that $w = -v$. The voltage across the capacitor remains at the value it had when the diode became reverse biased, and the voltage across the diode holds at the negative of this, which will keep the diode reverse biased.

Notice that the voltage $u$ across the inductor abruptly drops to zero. This is not physically troubling, however, because the current through the inductor at that time is zero.

The dependency relationships between variables in the two modes shown in Figure 33 are different. In the forwardBiased mode, $u$ has a direct dependency on $v$. In the reverseBiased mode, it does not. Figure 32 does not make this so explicit. One has to reason carefully about the diode model to infer these different relationships. Such differences, however, can significantly affect a simulation strategy, or even the validity of a simulation. We will illustrate this with a more complex example.

[6] Again, a more realistic diode model would all a non-zero leakage current when it is reverse biased. This is a trivial extension of this model.

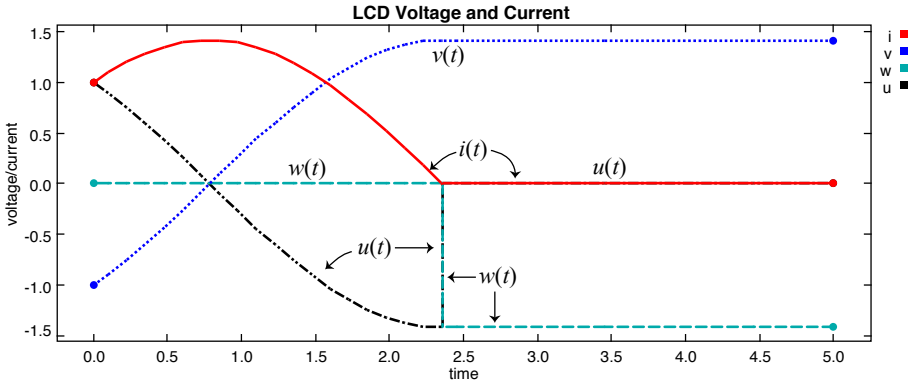


**Fig. 34.** Plot of an execution of the model in Figure 33.

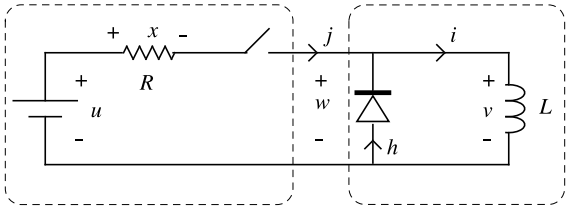


**Fig. 35.** A diode-inductor circuit from [40].

### 7.2 Diodes and Switches

The model in Figure 33 realizes a rather brute force strategy of constructing a separate model for each configuration of a circuit. This works fine when there is only one element in the circuit, the diode, that results in multi-modal behavior. What if there are more such elements?

Consider the circuit in Figure 35, which was studied by Mosterman and Biswas [40]. Such circuits are commonly used to protect inductive loads (such as motors) from high voltages that can result from being abruptly disconnected from a power source. The power source is at the left of the circuit, and the inductive load at the right. In normal operation, when the switch is closed, the diode is reverse biased, so no current (or a small leakage current) flows through it. When the switch is disconnected, if the diode is not present, the current through the inductor abruptly and discontinuously falls to zero. Equation (18) indicates that a very large (or in the ideal, infinite) negative voltage will be induced across the inductor. The role of the diode is to limit this voltage to prevent damage. Such a diode is often called a "**flyback diode**."

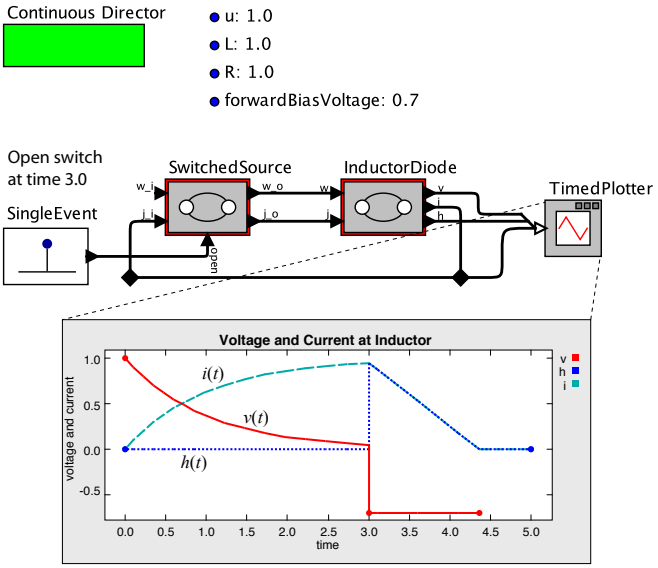


**Fig. 36.** A causal model for the circuit in Figure 35. [online]

This circuit has two multi-modal elements, a switch and a diode. Assuming that each of these has two modes, the number of modes of the circuit is the product of the number of modes of the elements. In order to anticipate larger circuits with more multi-modal components, we will want to construct a more modular model than one consisting of a distinct model for each mode of the overall circuit.

The figure shows dashed lines around two sub circuits, each of which has one dual-mode component. The interface between these sub circuits is a current $j$ and a voltage $w$. But the causal relationships are not clear, and in this case will depend on the modes of the sub circuits.

The top level of a causal model for this circuit is shown in Figure 36. The model has four parameters, the voltage $u$ applied by the ideal voltage source, the inductance $L$, the resistance $R$, and the voltage drop of a forward biased diode. With the parameters as shown in the figure, the model exhibits the plot shown in the figure. Initially, the switch is closed and no current $i$ is flowing through the inductor, so the voltage across the inductor is $u$ and the diode is reverse biased. Over time, the current $i$ increases and voltage $v$ drops. At time 3.0, the SingleEvent actor issues a discrete command to open the switch. This causes the diode to abruptly become reverse biased, imposing a fixed voltage of $-0.7$ volts (the forward bias voltage) across the inductor. Since the rate of change of current in an inductor is proportional to voltage (equation 18), the current $i$ (and $h$, the current through the diode) falls linearly. When that current hits zero, the diode ceases to be forward biased, and the current remains at zero.

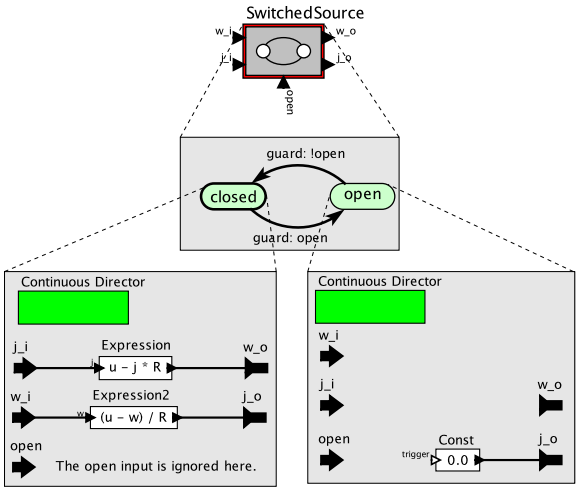


**Fig. 37.** Implementation of the SwitchedSource in Figure 36.

In Figure 36, the component labeled "SwitchedSource" represents the left side of the circuit in Figure 35. Its implementation is shown in Figure 37. Notice in particular that in this implementation, the current $i$ and the voltage $w$ are provided both as inputs and outputs. This allows the component to be used in a variety of ways. In Figure 36, the current input $j$ is provided, but not the voltage input $w$. Because this modeling framework is rooted in synchronous-reactive models, absence of an input is a well-defined concept, so the lack of a voltage input will simply result in the lack of the current output.

The SwitchedSource has two modes, open and closed. When the switch is closed, if the current through the load is provided as an input, then the output voltage to the load will be provided. It is calculated by calculating the voltage drop across the resistor using the standard formula for a **resistor**,

$$x = jR. \tag{19}$$

Hence, if the current $j$ is provided as an input, the output voltage will be

$$w = u - x = u - jR.$$

If instead the voltage $w$ is provided as an input, then the current $j$ will be given by

$$j = x/R = (u - w)/R.$$

When the switch is open, the inputs are ignored, and a constant zero output current is provided. Notice that the causal relationships between inputs and output depend on the mode.

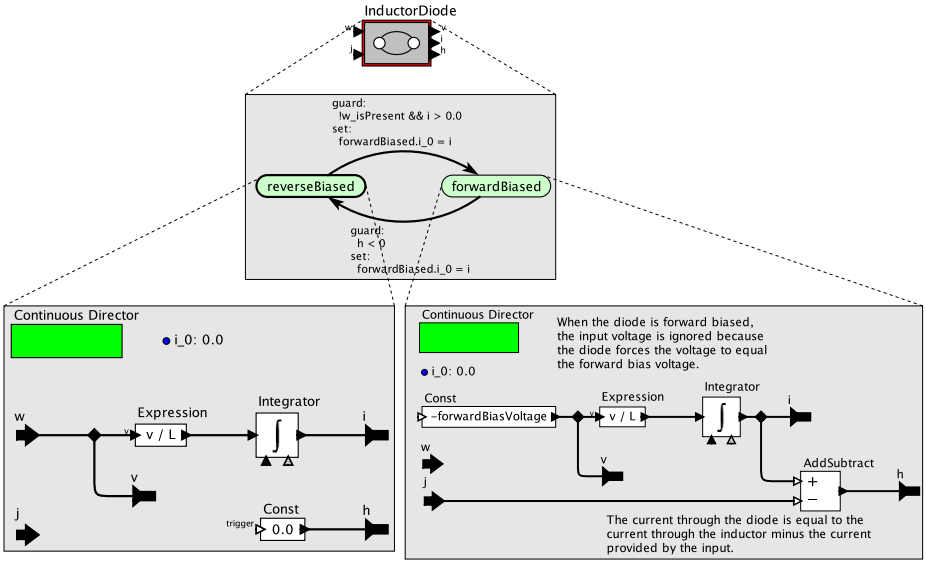


**Fig. 38.** Implementation of the InductorDiode in Figure 36.

In Figure 36, the component labeled "InductorDiode" represents the right side of the circuit in Figure 35. Its implementation is shown in Figure 38. When the diode is reverse biased, the input is the voltage across the diode, which is equal to the voltage across the inductor. The output is the current through the inductor.

In this mode, the input current $j$ is ignored. In fact, in this mode, it is physically problematic to provide an input $j$. An inductor stores current, so if the current through the inductor is an input to the model, then the storage element has to be abruptly and discontinuously modified. This would be an impulsive change, and could be realized using the same (problematic) mechanisms explored in Section 7.4 below. But this model does not need such mechanisms, and in fact, any attempt to use them would probably result in a causality loop.

If no input $w$ is provided, the output voltage $i$ will be absent. This explains a subtle point about the plot in Figure 36, where the voltage $v$ becomes absent around time 4.4. This is a modeling choice, of course. It is a simple exercise to change the model to assert a zero output voltage in the reverseBiased mode when the input $w$ is absent.

Let us carefully go through what occurs at time 3.0, when the switch is opened. First, at superdense time $(3,0)$, the SwitchedSource is in mode closed and InductorDiode is in mode reverseBiased. The output current $i$ from InductorDiode is the input to the SwitchedSource, which responds by providing an input voltage $w$ back to InductorDiode. All of this occurs simultaneously and instantaneously at time $(3,0)$. It is a fixed point.

At superdense time $(3,1)$, the SingleEvent actor provides a value *true* to the input port of SwitchedSource labeled "open." The currents and voltages at this time $(3,1)$ will

all be the same as they were at $(3,0)$, because nothing has changed (yet). According to the operational semantics of modal models [29], a mode transition occurs after the fixed-point solution has been found. At superdense time $(3,2)$, the voltage output of SwitchedSource will be absent, because the SwitchedSource mode is open. At that time, InductorDiode is still in its reverseBiased mode, so its outputs $i$ and $v$ will also be absent. After the fixed-point has been found (where most signals are absent), InductorDiode will take a transition to forwardBiased. At superdense time $(3,3)$, both components are in their final modes, and the new fixed point represents the starting point for the next integration interval.

Note that the circuit model is at one superdense time instant in the open and reverseBiased modes. But it does not linger in this mode. Since it spends zero time in this mode, the signal values (mostly absent) that are the fixed point in this mode are harmless. They do not affect integrators that are keeping track of the physical state of the system. Such transitory modes are called "mythical modes" by Mosterman and Biswas [40].

### 7.3 Complexity

Notice that in a circuit with two elements, each of which has two modes, there are a total of four modes. One modeling approach would be to construct a separate circuit model for each of the modes of the overall circuit, and to build the logic that switches between modes, carrying the state of the circuit while switching modes. A more complicated circuit, however, will have many more modes. In fact, the number of modes will grow exponentially with the number of modal elements. This complexity is intrinsic in such circuits, and suggests that the struggles that scientists and engineers have had modeling such circuits is not just a consequence of poor understanding of the physics. It is because these circuits are fundamentally complex, exhibiting an exponential number of distinctly different behaviors with complex transitions between these behaviors. Indeed, electrical systems with discrete elements can get extremely complicated [45]. Such complex examples would yield a truly vast number of modes in a flat multi-modal model.

The modeling style represented in Figure 36 is preferable. It does not eliminate the complexity, but it makes it more manageable and understandable.

### 7.4 Nonconstructive Circuits

Some circuits do not yield easily to reasonable solutions. Mosterman and Biswas consider a suite of circuits in [41] that are variants of Figure 39. Assume that the switch is initially open and the capacitors start with unequal voltages, $v(0) \neq u(0)$. When the switch is closed, the two voltages, by definition of a closed switch, become equal instantaneously. This requires an impulsive input to the capacitors to instantaneously alter their stored values.

A first attempt to construct an executable model for Figure 39 is shown in Figure 40. However, this model fails to execute because it has a causality loop, and hence is not constructive. The EnabledComposite will be triggered at the same superdense time where an enable event is provided by the SingleEvent actor. At this index, the least fixed

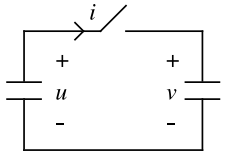


**Fig. 39.** Two capacitors and a switch.

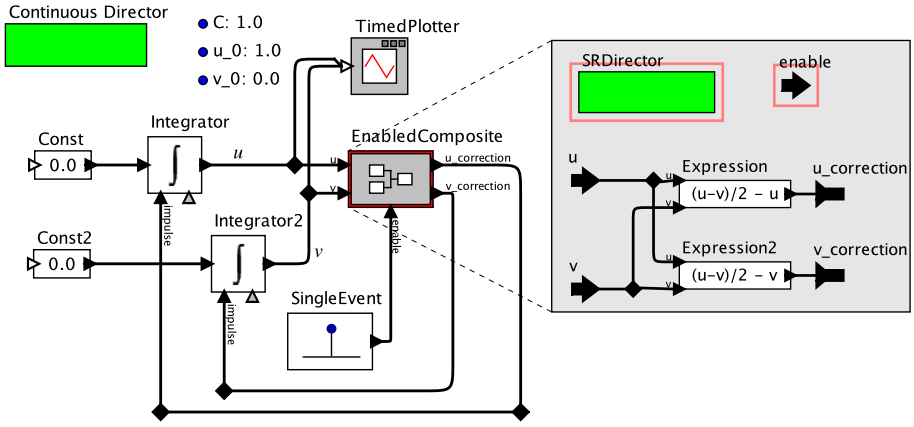


**Fig. 40.** Attempt at a model for Figure 39.

point results in unknown values for $u$ and $v$. This is because the integrators need to know their impulse inputs before they can assert an output value, and the EnabledComposite needs to know the values of $u$ and $v$ before it can provide the impulsive corrections to the integrators.

We can change the model to make it executable by manually adding a MicrostepDelay actor in the feedback paths, which will result an executable model providing the plot shown in Figure 41. In that figure, the switch is closed at time 1.0, and at that time, instantaneously, the voltages of the two capacitors equalize. But to what value?

As pointed out in [41], the charge stored in a capacitor is proportional to the voltage, but the energy stored in a capacitor is proportional to the voltage squared. The plot in Figure 41 conserves charge, but it instantaneously loses energy. But there is no dissipative element in the model that can lose energy, so the model appears to correlate to flawed physics. Perhaps this is related to the time-energy uncertainty of [37]. So while our attempt to circumvent a causality loop results in a model that can be executed, the model is suspect, just as the multi-body collision models above are suspect.

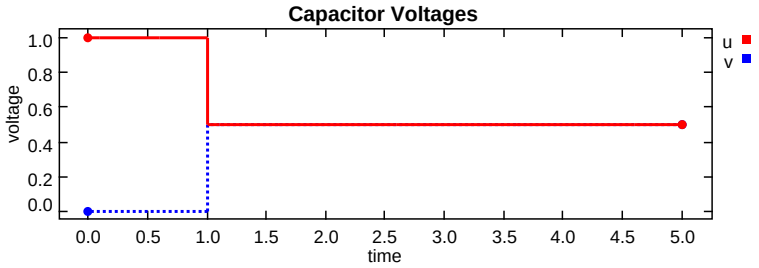


**Fig. 41.** Execution of a variant of the model in Figure 40 where a MicrostepDelay has been inserted in each feedback path. This execution conserves charge but not energy. [online]

## 8 Related Work

Superdense time was introduced in [35] and [38] and subsequently used for modeling hybrid systems in [24] and [31], and also in [23], where it was called "hyper time." The trace semantics and interleaving semantics of Alur and Henzinger [2] also have the flavor of superdense time, since multiple ordered events can be simultaneous in time. The hardware description language VHDL has a related model of time, where time is a member of $\mathbb{N} \times \mathbb{N}$, and the second value is used in a manner similar to the superdense time index [4]. This paper adopts the approach in [32], where at each instant in superdense time, a model has a fixed-point semantics.

In this paper, models are constructed with causal components, which have explicit inputs and outputs. An alternative is so-called **equational languages**, where the interfaces between components are neither inputs nor outputs, but rather assert equality. Bond graphs [49], Modelica [57], and SPICE [43] all have this character. They emphasize a style of modeling that focuses on describing systems directly in terms of conservation laws, such as equations (5) and (6), rather than input/output relationships. Figure 32, for example, has three interconnected acausal components. Figure 33 has translated the entire circuit into a causal model.

The Ptolemy II framework used in this paper, in contrast, uses a style similar to other popular modeling languages such as Simulink and LabVIEW. As illustrated in this paper, a person building a model in such a language may have to puzzle over which variables to define as inputs and which to define as outputs of which components. If the model builder is constructing the model in an acausal language, then the *compiler* has to construct a causal variant in order to simulate the model. The compiler will face the same problems.

Acausal languages such as Modelica allow system dynamics to be given using differential algebraic equations (DAEs), which are more general than ODEs. DAEs include equations that specify algebraic relationships between variables, where "algebraic" simply means here that the relationships do not involve integration. Such relations can form cycles, in that $x$ depends on $y$ and vice versa, but these cycles do not automatically lead

to causality loops or nonconstructive models. In particular, given a system of DAEs, a Modelica compiler will attempt to remove apparent causality loops through a process known as **index-order reduction**, using for example the Pantelides algorithm [48]. When this process is successful, causality loops have been removed, and the resulting models resemble the ODE models that we use in this paper.

In many cases, the acausal representation is more compact and readable. However, such a representation may mask nondeterminacy or complexity, leading a designer into a false sense of confidence. Moreover, acausal representations are not always more intuitive or easier to understand. Consider a pendulum for example. To this author, a causal model in the transformed polar coordinate system is easier to construct and understand than the standard acausal model.

This paper is not taking a position about whether it is preferable to *specify* models using causal or acausal languages, but instead is showing how such models need to be executed. Whether they are specified as shown in Figure 32 and then translated into Figure 33, or specified directly as in Figure 33, is irrelevant to this paper.

Similarly, in this paper, all models are given in a visual block diagram notation. This is useful for pedagogical purposes, but this paper is not taking a position on whether such visual representations are to be preferred over textual languages. Some modeling languages, such as Modelica, provide both visual and textual notations, a recognition that each has its place. Which notation is used is again irrelevant to this paper.

Modal models have appeared in many forms in the literature, perhaps beginning with Statecharts [22]. This paper assumes the semantics in [29]. Modelica also supports modal behaviors [52,47]. Sztipanovits et al. also describe modal models [56]. Hybrid Bond graphs are also modal models [41].

The constructive semantics is due to Berry [9], who applied it to the Esterel programming language and to circuit analysis [53]. Recently, Mendler et al. have proven that non-constructive circuit models correspond to physical systems that are vulnerable to unstable oscillation [39].

## 9 Conclusion

Constructive semantics gives a natural way to separate problems that can be solved with confidence from those that cannot. When, for example, the order of nearly instantaneous collisions is important, a constructive semantics forces us to either choose an order or explicitly choose nondeterminism. Moreover, given a nonconstructive and a constructive model of the same physical phenomenon, the constructive one is more like to be useful and faithful to the physics than the nonconstructive one. Building useful constructive models of combined continuous and discrete behaviors is facilitated by a superdense model of time, an explicit use of impulses (generalized functions), and modal models.

## 10 Acknowlegements

Thanks to Gérard Berry, David Broman, Lev Greenberg, Pieter Mosterman, Marc Pouzet, Stavros Tripakis, and Michael Wetter for extensive discussions that led to this paper.

And thanks to my former students, Adam Cataldo, Eleftherios Matsikoudis, Jie Liu, Xiaojun Liu, and Haiyang Zheng, for teaching me about continuous-time models.

## References


1. ALUR, R., COURCOUBETIS, C., HALBWACHS, N., HENZINGER, T., HO, P.-H., NICOLLIN, X., OLIVERO, A., SIFAKIS, J., AND YOVINE, S. The algorithmic analysis of hybrid systems. *Theoretical Computer Science 138*, 1 (1995), 3–34.
2. ALUR, R., AND HENZINGER, T. Logics and models of real time: A survey. In *REX Workshop* (Mook, The Netherlands, 1991), J. W. De Bakker, C. Huizing, W. P. De Roever, and G. Rozenberg, Eds., vol. LNCS 600, Springer, pp. 74–106.
3. ANDRÉ, C. SyncCharts: A visual representation of reactive behaviors. Tech. Rep. RR 95–52, rev. RR (96–56), I3S, April 1996.
4. ARMSTRONG, J. R., AND GRAY, F. G. *VHDL Design Representation and Synthesis*, second ed. Prentice-Hall, 2000.
5. BEER, JR., F., AND JOHNSTON, E. R. *Vector equations for Engineers: Dynamics*, sixth edition ed. McGraw Hill, 1996.
6. BENVENISTE, A., AND BERRY, G. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE 79*, 9 (1991), 1270–1282.
7. BENVENISTE, A., BOURKE, T., CAILLAUD, B., AND POUZET, M. The fundamentals of hybrid systems modelers. *Journal of Computer and System Sciences 78*, 3 (2012), 877–910.
8. BENVENISTE, A., BOURKE, T., CAILLAUD, B., AND POUZET, M. Index theory for hybrid DAE systems (abstract and slides). In *Synchronous Programming (Synchron)* (Schloss Dagstuhl, Germany, 2013), vol. 13471, Dagstuhl.
9. BERRY, G. *The Constructive Semantics of Pure Esterel*, draft version 3 ed. Unpublished, 1999.
10. BLIUDZE, S., AND FURIC, S. An operational semantics for hybrid systems involving behavioral abstraction. In *Modelica Conference* (Lund, Sweden, 2014).
11. BOX, G. E. P., AND DRAPER, N. R. *Empirical Model-Building and Response Surfaces*. Wiley Series in Probability and Statistics. Wiley, 1987.
12. CARDOSO, J., LEE, E. A., LIU, J., AND ZHENG, H. Continuous-time models. In *System Design, Modeling, and Simulation using Ptolemy II*, C. Ptolemaeus, Ed. Ptolemy.org, Berkeley, CA, 2014.
13. CATALDO, A., LEE, E. A., LIU, X., MATSIKOUDIS, E., AND ZHENG, H. A constructive fixed-point theorem and the feedback semantics of timed systems. In *Workshop on Discrete Event Systems (WODES)* (Ann Arbor, Michigan, 2006).
14. CHATTERJEE, A., AND RUINA, A. A new algebraic rigid body collision law based on impulse space considerations. *Journal of Applied Mechanics 65*, 4 (1998), 939–951.
15. DAVEY, B. A., AND PRIESTLY, H. A. *Introduction to Lattices and Order*, second ed. Cambridge University Press, 2002.
16. EDWARDS, S. A., AND LEE, E. A. The semantics and execution of a synchronous block-diagram language. *Science of Computer Programming 48*, 1 (2003), 21–42.
17. ERLEBEN, K., SPORRING, J., HENRIKSEN, K., AND DOHLMANN, H. *Physics-Based Animation*. Charles River Media, Inc., Hingham, MA, 2005.
18. FENG, T. H., LEE, E. A., LIU, X., TRIPAKIS, S., ZHENG, H., AND ZHOU, Y. Modal models. In *System Design, Modeling, and Simulation using Ptolemy II*, C. Ptolemaeus, Ed. Ptolemy.org, Berkeley, CA, 2014.
19. GLOCKNER, C. Scalar force potentials in rigid multibody systems. In *Multibody Dynamics with Unilateral Contacts*, F. Pfeiffer and C. Glockner, Eds., vol. 421 of *CISM Courses and Lectures*. Springer, Vienna, New York, 2000.

20. GOLOMB, S. W. Mathematical models: Uses and limitations. *IEEE Trans. on Reliability R-20*, 3 (1971), 130–131.
21. HALBWACHS, N., CASPI, P., RAYMOND, P., AND PILAUD, D. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE 79*, 9 (1991), 1305–1319.
22. HAREL, D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming 8* (1987), 231–274.
23. IWASAKI, Y., FARQUHAR, A., SARASWAT, V., BOBROW, D., AND GUPTA, V. Modeling time in hybrid systems: how fast is instantaneous'? In *International Workshop on Qualitative Reasoning* (Amsterdam, 1995), Morgan Kaufmann, pp. 1773–1780.
24. KAPUR, A. *Interval and Point-Based Approaches to Hybrid Systems Verification*. Ph.d., Stanford University, 1997.
25. LEE, E. A. Modeling concurrent real-time processes using discrete events. *Annals of Software Engineering 7* (1999), 25–45.
26. LEE, E. A., NEUENDORFFER, S., AND WIRTHLIN, M. J. Actor-oriented design of embedded hardware and software systems. *Journal of Circuits, Systems, and Computers 12*, 3 (2003), 231–260.
27. LEE, E. A., AND SANGIOVANNI-VINCENTELLI, A. A framework for comparing models of computation. *IEEE Trans. on Computer-Aided Design of Circuits and Systems 17*, 12 (1998), 1217–1229.
28. LEE, E. A., AND SESHIA, S. A. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. LeeSeshia.org, Berkeley, CA, 2011.
29. LEE, E. A., AND TRIPAKIS, S. Modal models in Ptolemy. In *3rd Int. Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT)* (Oslo, Norway, 2010), vol. 47, Linköping University Electronic Press, Linköping University, pp. 11–21.
30. LEE, E. A., AND VARAIYA, P. *Structure and Interpretation of Signals and Systems*, 2.0 ed. LeeVaraiya.org, 2011.
31. LEE, E. A., AND ZHENG, H. Operational semantics of hybrid systems. In *Hybrid Systems: Computation and Control (HSCC)* (Zurich, Switzerland, 2005), M. Morari and L. Thiele, Eds., vol. LNCS 3414, Springer-Verlag, pp. 25–53.
32. LEE, E. A., AND ZHENG, H. Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems. In *EMSOFT* (Salzburg, Austria, 2007), ACM, pp. 114 – 123.
33. LINDSTRØM, T. An invitation to nonstandard analysis. In *Nonstandard Analysis and Its Applications*, N. Cutland, Ed. Cambridge University Press, 1988, pp. 1–105.
34. LIU, J., AND LEE, E. A. On the causality of mixed-signal and hybrid models. In *6th International Workshop on Hybrid Systems: Computation and Control (HSCC '03)* (Prague, Czech Republic, 2003).
35. MALER, O., MANNA, Z., AND PNUELI, A. From timed to hybrid systems. In *Real-Time: Theory and Practice, REX Workshop* (1992), Springer-Verlag, pp. 447–484.
36. MALIK, S. Analysis of cyclic combinational circuits. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 13*, 7 (1994), 950–956.
37. MANDELSHTAM, L. I., AND TAMM, I. The uncertainty relation between energy and time in nonrelativistic quantum mechanics. *Journal of Physics IX*, 4 (1945), 249–254.
38. MANNA, Z., AND PNUELI, A. Verifying hybrid systems. In *Hybrid Systems* (1993), vol. LNCS 736, pp. 4–35.
39. MENDLER, M., SHIPLE, T. R., AND BERRY, G. Constructive boolean circuits and the exactness of timed ternary simulation. *Formal Methods in System Design 40*, 3 (2012), 283–329.
40. MOSTERMAN, P. J., AND BISWAS, G. Modeling discontinuous behavior with hybrid bond graphs. In *Ninth Qualitative Reasoning Workshop* (Amsterdam, 1995), pp. 139–147.

41. MOSTERMAN, P. J., AND BISWAS, G. A theory of discontinuities in physical system models. *Journal of the Franklin Institute 335*, 3 (1998), 401–439.
42. MOSTERMAN, P. J., SIMKO, G., AND ZANDE, J. A hyperdense semantic domain for discontinuous behavior in physical system models. In *Multi-Paradigm Modeling (MPM)* (2013).
43. NAGEL, L. W., AND PEDERSON, D. O. SPICE (simulation program with integrated circuit emphasis). Technical memorandum, Electronics Research Laboratory, University of California, Berkeley, April 1973.
44. NORTON, J. D. Causation as folk science. In *Causation, Physics, and the Constitution of Reality*, H. Price and R. Corry, Eds. Clarendon Press, Oxford, 2007, pp. 11–44.
45. NUZZO, P., XU, H., OZAY, N., FINN, J. B., SANGIOVANNI-VINCENTELLI, A. L., MURRAY, R. M., DONZE, A., AND SESHIA, S. A. A contract-based methodology for aircraft electric power system design. *IEEE Access 2* (2014), 1–25.
46. OTTER, M., ELMQVIST, H., AND LÓPEZ, J. D. Collision handling for the Modelica multibody library. In *Modelica Conference* (Hamburg, Germany, 2005), pp. 45–53.
47. OTTER, M., MALMHEDEN, M., ELMQVIST, H., MATTSSON, S. E., AND JOHNSSON, C. A new formalism for modeling of reactive and hybrid systems. In *Modelica Conference* (Como, Italy, 2009), The Modelica Association.
48. PANTELIDES, C. C. The consistent initialization of differential-algebraic systems. *SIAM Journal of Scientific and Statistical Computing 9*, 2 (1988), 213–231.
49. PAYNTER, H. M. *Analysis and Design of Engineering Systems*. The M.I.T. Press, Cambridge, MA, 1961.
50. PRICE, H., AND CORRY, R., Eds. *Causation, Physics, and the Constitution of Reality*. Clarendon Press, Oxford, 2007.
51. PTOLEMAEUS, C., Ed. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, Berkeley, CA, 2014.
52. SCHAMAI, W., POHLMANN, U., FRITZSON, P., PAREDIS, C. J., HELLE, P., AND STROBEL, C. Execution of UML state machines using Modelica. In *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT)* (Oslo, Norway, 2010), vol. 47, Linköping University Electronic Press, Linköping University, pp. 1–10.
53. SHIPLE, T. R., BERRY, G., AND TOUATI, H. Constructive analysis of cyclic circuits. In *European conference on Design and Test (DATE)* (1996), IEEE Computer Society, pp. 328–333.
54. STEWART, D. E. Rigid-body dynamics with friction and impact. *SIAM Review 42*, 1 (2000), 3–39.
55. STRONGE, W. J. Rigid body collisions with friction. *Proceedings of the Royal Society of London 431* (1990), 169–181.
56. SZTIPANOVITS, J., WILKES, D. M., KARSAI, G., BIEGL, C., AND LESTER E. LYND, J. The multigraph and structural adaptivity. *IEEE Trans. on Signal Processing 41*, 8 (1993), 2695–2716.
57. TILLER, M. M. *Introduction to Physical Modeling with Modelica*. Kluwer Academic Publishers, 2001.