S

Simulation of Hybrid Dynamic Systems

Pieter J. Mosterman, Akshay Rajhans, Anastasia Mavrommati, and Roberto G. Valenti
Advanced Research & Technology Office, The MathWorks, Inc., Natick, MA, USA

Abstract

Hybrid dynamic systems combine continuous and discrete behavior. An overview of semantic domains of dynamic behavior is presented in terms of the evolution domain, the value domain, and the execution domain. Simulation of piecewise continuous behavior interspersed with discontinuities is investigated in detail and illustrated by an example that motivates the use of simulation support for hyperdense time as an evolution domain. Engineered systems are characterized by combinations of compute elements, network connectivity, sensors and actuators, and a physical component. The various different elements benefit from modeling formalisms based on various semantic domains. Efficient execution is achieved by a combination of time-driven and event-driven execution engines.

Keywords

Cyber-Physical Systems · Dynamic Systems · Hybrid Systems · Modeling and Simulation · Modeling Physics · Mulitparadigm Modeling · Numerical Simulation

Introduction

Hybrid dynamic systems combine continuous and discrete dynamics. In the literature, hybrid dynamic systems are often referred to simply as hybrid systems where system is to be taken as a mathematical system (as opposed to a physical system). Hybrid systems as a term, however, are also used to refer to systems such as a hybrid powertrain where a combustion engine and electric motor connect to the driveline. Therefore, to avoid confusion, the term hybrid dynamic systems is used.

Hybrid dynamic systems enable a selective choice in how to represent modeled phenomena. Modeling a system under study by hybrid dynamics allows improved efficiency in terms of simulation. Moreover, the ability to include behavior in a coarse sense prevents the need for detailed parameter knowledge. For example, when an electrical switch is opened, its current flow does not fall to 0 instantaneously, but the opening creates a quickly decreasing contact area that results in a quickly decreasing change in current. Moreover, a quickly increasing air gap may support a brief moment of current flow with a corresponding spark. The details of these effects (the properties of the contact area and the air gap as they change during a brief period of time)

J. Baillieul, T. Samad (eds.), *Encyclopedia of Systems and Control*,
https://doi.org/10.1007/978-1-4471-5102-9_100048-2

may be unknown and difficult to measure while being irrelevant to the overall behavior of interest (the current changes to 0). Other examples of physical components that may be well modeled by discrete on/off type behavior include diodes, valves, latches, clutches, relays, and so forth. Likewise, components that effect quick changes compared to the overall behavior of interest such as an analog to digital converter may be modeled by discontinuous change in their behavior.

Even though systems that include components with on/off or more generally discontinuous behavior may be intuitively modeled as hybrid dynamic systems (e.g., an antilock braking system, a rectifier circuit, an overflowing tank), a system under study is not intrinsically a hybrid dynamic system. A model is a chosen representation of a system under study depending on a specific purpose. For example, to achieve real-time simulation, a model that abstracts fast temporal behavior into discontinuities oftentimes is preferred. However, to gain insight in the physics of a system, it is conceptually preferred to capture fast behavior in detail (Breedveld 1996).

A key aspect of being a representation is that a model must be simpler than the system under study. Thus, a perfect copy would not be considered a model. This notion may be confounded with the model being "wrong," yet if the model serves its purpose, it helps solve a problem, it is in fact "right." The quality of the model reflects in its efficiency to solve the problem. Combining continuous dynamics with discrete dynamics provides flexibility in modeling to include the detail necessary but no more (conform the principle of parsimony or Occam's razor).

Models of hybrid dynamic systems can be studied and leveraged in multiple ways, for example, formal property proving to guarantee the absence of design errors or automatic generation of embedded code. The focus of this entry, however, is on numerical simulation of such models, that is, given a hybrid system model and the initial conditions, numerically simulating the execution of the dynamics on a computer. Such an execution of hybrid dynamic systems requires support for intricacies that bring to bear a unique set of challenges beyond the execution of continuous and discrete dynamics alone. In the proceeding, a number of motivating examples are presented to introduce various different forms of hybrid dynamic systems. Next, a range of semantic domains for dynamic systems and their execution semantics are introduced. The class of piecewise continuous systems is then discussed in detail as it embodies the broadest range of behaviors specific to hybrid dynamic systems. Next, the utility of the various different semantic domains in multiparadigm modeling is sketched followed by a discussion of execution engines to achieve efficient behavior generation. Finally, conclusions and open challenges are presented.

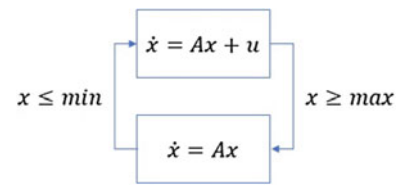
Motivating Examples

Examples of hybrid dynamic systems often used in the literature include a thermostat, a bouncing ball, and a freewheeling diode. Each of these allows concentrating on a particular aspect of hybrid behavior, namely, a discontinuous forcing function, state reinitialization, and event iteration, respectively.

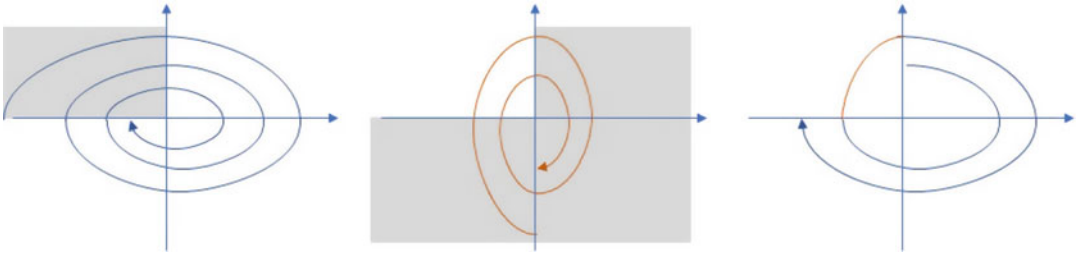
Switched Control: A Thermostat

Switched control is a popular approach, for example, for tackling nonlinear system behavior. Piecewise linearization by approximating or abstracting the nonlinear behavior using a number of different modes of linear behavior allows synthesis methods to find control laws in each of the modes. Switching logic then selects the corresponding mode as the system under control (the process or plant) evolves.

Another example is a thermostat that employs bang-bang control. Here, the actuation is at its



Simulation of Hybrid Dynamic Systems, Fig. 1 Thermostat model. © The MathWorks, Inc.



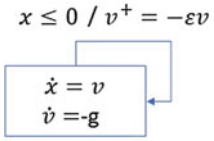
Simulation of Hybrid Dynamic Systems, Fig. 2 Combining asymptotically stable behavior leads to unstable behavior. © The MathWorks, Inc.

maximum (maximum heating) or its minimum (no heating). Figure 1 shows a hybrid dynamic system model of a thermostat as a finite state machine with a differential equation that models the behavior in each discrete state. The thermal behavior of the room is represented by the differential equation $\dot{x} = Ax + u$ with x being the temperature and u the heating control input. The heating u is active or not, depending on the state. When the temperature falls below a minimum ($x \leq min$), the heating turns on, and when the temperature exceeds a maximum ($x \geq max$), the heating turns off.

From a simulation perspective, there are two important considerations: (i) does the model always remain in one of the states for a duration of time (i.e., is $max > min$) and (ii) is it necessary to reset the numerical solver that generates behavior for the differential equations? The second concern depends on the solver type, where a multistep solver exploits computed values of previous time steps and assumes a certain order of continuity between time steps (Cellier and Kofman 2006).

While the simulation considerations are relatively straightforward, from a control perspective, hybrid behavior may evidence problematic behavior. For example, Fig. 2 shows two asymptotically stable behaviors, one on the left-hand side and one in the center. When switched control selects the behavior in the top-left quadrant of the center behavior to produce the combined behavior on the right-hand side, the resultant behavior becomes unstable. Issues of stability, reachability, nondeterminism, and so forth are particularly difficult in case of hybrid dynamic systems but beyond the scope of the simulation focus.

Simulation of Hybrid Dynamic Systems, Fig. 3 Bouncing ball model. © The MathWorks, Inc.



State Reinitialization: A Bouncing Ball

A key behavior to support in simulation of hybrid dynamic systems is the reinitialization of states in differential equations (the continuous state). Figure 3 shows a model of a bouncing ball, a much studied example of this behavior. The differential equations that are active in the discrete state capture the velocity of the ball, v, as subject to the gravitational acceleration, g. When the position of the ball, x, falls below 0 (to represent the floor level), an elastic collision sets the velocity of the ball to a new value v^+ based on the velocity upon collision and a restitution coefficient ε.

The discontinuity as introduced by reinitializing the continuous state requires the same numerical solver considerations as the discontinuity introduced by switching input of the thermostat problem. The bouncing ball, however, highlights detecting the collision event as a challenge. Consider ε to be positive but less than 1. As the ball bounces repeatedly, the maximum height of each bounce progressively decreases. When this maximum height becomes less than the numerical tolerance set to detect whether $x \leq 0$, the

transition is always enabled. At this point, further simulation becomes ill defined.

Event Iteration: A Freewheeling Diode

A key feature of hybrid dynamic systems is the occurrence of a sequence of discrete state changes with no duration of continuous-time behavior in between. Figure 4a shows an electrical circuit with two elements modeled with discontinuous behavior: (i) the switch, S, enforces a 0 voltage drop when closed and a 0 current flow when open, and (ii) the diode, D, enforces a 0 voltage drop when on and a 0 current flow when off. When the switch is closed, the battery, B, supplies a voltage that builds up a flux in the inductor, L. The corresponding current flow creates a voltage drop across the resistor, R, and when less than the battery voltage, this results in a positive voltage across the diode in its off state.

Figure 4b shows a model of the electrical circuit as a state machine with three parts. At the top, the equations for the inductor are shown where the inductor flux, p, builds up based on the voltage drop, V_D, across the parallel branch. The current through the switch, I_S, minus the current through the diode, I_D, is the inductor current and directly represented as the flux per inductance, $\frac{p}{L}$.

Below the equations are two state machines, one to model the behavior of the switch and one for the diode. The switch is opened when time, t, reaches 1. At the point in time when the condition $t \geq 1$ becomes true, the switch changes its behavior from allowing arbitrary current with 0 voltage drop to allowing an arbitrary voltage with 0 current flow. Given that the diode also enforces 0 current flow, the inductor current is forced to 0, which requires an instantaneous discharge of its flux, p.

Because of the discontinuous change in flux, according to the differential equation $V_D = \dot{p}$, a negative voltage spike across the parallel branch would occur. This voltage, however, enables the $V_D \geq 0$ condition (notice the orientation of the diode with its positive port connected to ground), and the diode comes on such that the diode provides the inductor current instead of forcing it to 0.

In addition to the complexity of handling such event sequences, hybrid dynamic system simulation must handle impulsive behavior such as the voltage spike. Moreover, the simulation must be able to compute discontinuous changes in continuous state from a system of equations (i.e., the reinitialization value is not explicitly provided as an assignment).

Semantic Domains

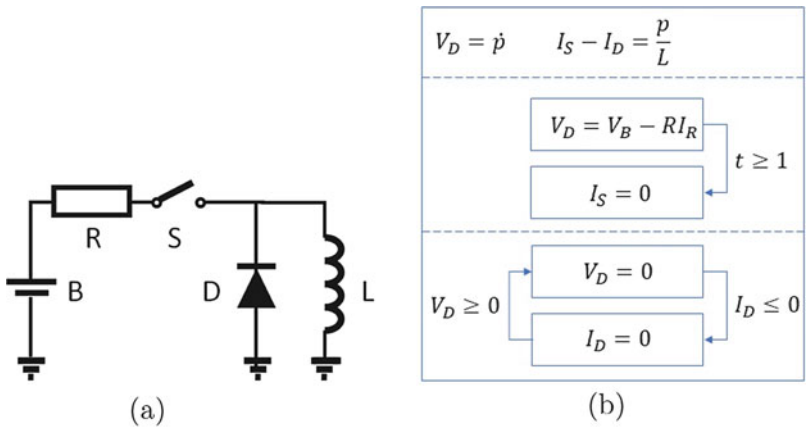
Given the notion that hybrid dynamic systems combine continuous and discrete behavior, it is instructive to investigate in more detail the semantic domains used in modeling and simulation. To establish a fundamental structure of these semantic domains, note that the dynamic nature of the systems implies there is a domain on which behavior evolves. A behavior, β, is then a mapping of an evolution to values where a hybrid dynamic system can be thought of as a partial function with the evolution domain, $\mathcal{E}$, as its domain and the value domain, $\mathcal{V}$, as its codomain

$$\beta : \mathcal{E} \mapsto \mathcal{V} \tag{1}$$

In addition to the evolution and value domains, dynamic systems often allow value changes at specific instances only. These instances form the execution domain, $\mathcal{X}$.

Integer Evolution Behavior

In its basic form, an evolution domain comprises a set of linearly ordered instances. This can be represented by the natural numbers, $\mathbb{N}$, even if the metric notion may not be relevant. For example, program code such as C, Java, and MATLAB® (MathWorks® 2019), when single threaded, follows a linear sequence of operations that can be indexed by integers. Execution of each operation follows completion of the previous operation, and so a program counter progresses along the integers without the need for independent dynamic scheduling. The discrete evolution domain is depicted in Fig. 5a by a dashed horizontal axis, while the execution ordering is represented by squared arrows between



Simulation of Hybrid Dynamic Systems, Fig. 4 Freewheeling diode. (**a**) Circuit diagram. (**b**) Dynamics model. © The MathWorks, Inc.

consecutive execution instances. The solid vertical arrow representing the value domain captures the continuous values of computational operations. The solid circles depict values computed at discrete execution instances. It should be acknowledged that computational operations approximate continuous values by floating point representations, and so the domain is not truly continuous. In the value domain, this distinction is not important to the discussion.

Time is an evolution domain that is integral to hybrid dynamic systems. In computational implementations, it is often preferred to discretize time in periodic steps of equal size, the sample time. Figure 5b depicts time as a discrete evolution domain such that it can be specified by difference equations or synchronous data flow (Benveniste et al. 2003). Note that the base rate of the scheduler has a sample time that is represented by the vertical lines, but not every base rate sample hit corresponds to the execution of an operation. For example, in case of a system with operations that have a sample time of 2 s and operations that have a sample time of 3 s, the base rate has a sample time of 1 s (the greatest common divisor). At 5 s, the base rate has a sample hit but neither of the other sample times does. As in programming code, the value domain is continuous.

Because of the integer nature of the execution, in case there is no real-time execution requirement, the scheduler may operate similar to the execution of program code and progress along an integer schedule. The sample times of operations are harmonics of the base rate and, therefore, execute at integer multiples of the execution index. In case real-time behavior is required, each integer advance is invoked based on a periodic interrupt from the compute platform.

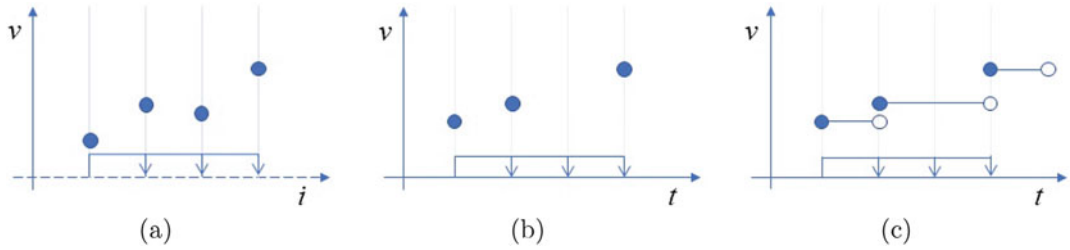
Similar in execution but different in behavior is the discrete-time evolution domain shown in Fig. 5c. Here, the value of a behavior changes at a periodic rate, but the value is held and accessible in between sample times. This is indicated as a continuous evolution domain by the solid horizontal arrow. For digital control systems, the zero-order hold (ZOH) approach is often preferred, whereas for digital image processing systems, a sampled time approach is preferred because it does not affect the signal frequencies.

S

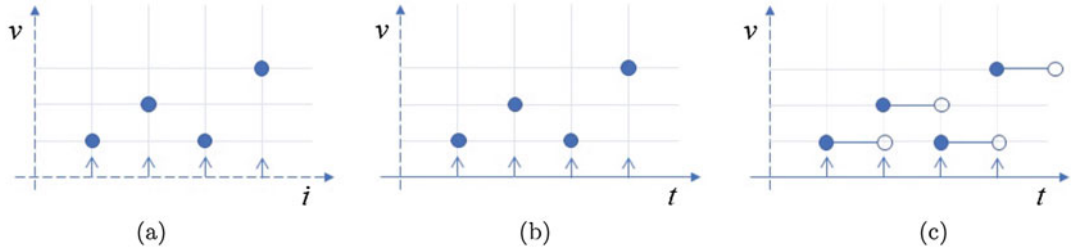
Finite State Value Behavior

Finite state transition systems behave similar to the integer execution of general computations in Fig. 5. The corresponding semantic domains are shown in Fig. 6 where the finite state notion is depicted as a discrete value domain by the dashed vertical coordinate axis. Note that even though in a computational implementation the states of finite state transition systems are typically identified by integers, semantically the discrete values that represent states are not ordered and are without a metric.

Figure 6a illustrates the semantic domain of transition systems (e.g., automata Alur and Dill (1994) and Petri nets Murata (1989)) based on



Simulation of Hybrid Dynamic Systems, Fig. 5 Integer evolution semantic domains. (**a**) Program code. (**b**) Sampled time. (**c**) Discrete time. © The MathWorks, Inc.



Simulation of Hybrid Dynamic Systems, Fig. 6 Discrete value semantic domains. (**a**) Transition system. (**b**) Sampled transition system. (**c**) Discrete-time transition system. © The MathWorks, Inc.

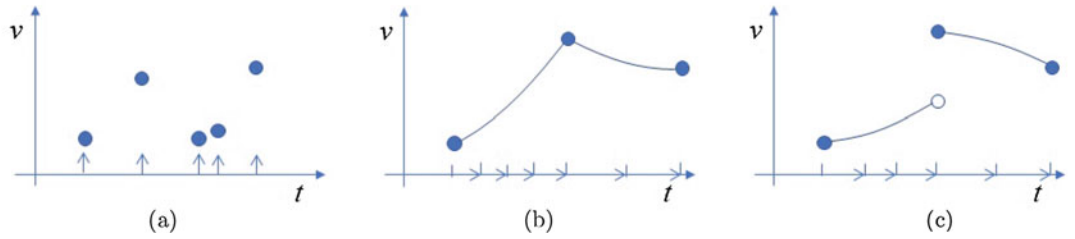
a discrete progression of discrete state changes. The states as well as the events that effect state changes can be indexed by integer values. In contrast to program code, the events are dynamically generated and not preassigned as the completion of a previous operation. For example, in a Petri net, the tokens in input places of a transition determine whether the transition is enabled. Thus, a scheduler dynamically schedules the transition to be active as the token distribution evolves. This dynamic execution behavior is depicted in Fig. 6 by the upward pointing arrows.

The semantic domains of the sampled and discrete-time versions of transition systems are illustrated in Figs. 6b, c, respectively. Associating time with the evolution domain enables sample times on a continuous domain, while the execution behavior is discrete and periodic. Given the dynamic scheduling nature, real-time behavior is not a determining factor whether a scheduler is necessary. In its execution, the sampled transition system corresponds to Mealy-type behavior where actions are associated with transitions. The discrete-time transition system corresponds to Moore-type behavior where actions are associated with states.

Continuous Execution Behavior

Executing events on a continuous schedule that is not periodic is the hallmark of discrete-event systems such as the discrete-event system specification (DEVS) (Zeigler et al. 2000). Figure 7a illustrates the semantic domain as a continuous evolution domain and a continuous value domain. Moreover, the discrete events at which value changes occur can be scheduled on a continuous execution domain. Note that transition systems with events occurring on a continuous domain would lead to a discrete value domain and bridge the semantic domains in Figs. 6b and 7a. However, in such transition systems, time itself is part of the state as clocks that can be reset, which creates a continuous value domain.

Behavior that is continuous in all its facets is illustrated in Fig. 7b. This behavior can be specified by switched differential equations, and the evolution domain in hybrid dynamic systems typically is time. Generating the behavior from the differential equations is based on gradients with respect to time. From an existing point in time, the gradients are used to extrapolate the behavior trace to a new point in time that approximates



Simulation of Hybrid Dynamic Systems, Fig. 7 Continuous evolution and value semantic domains. (**a**) Discrete-event system. (**b**) Continuous behavior. (**c**) Discontinuities. © The MathWorks, Inc.

the underlying solution of the differential equation within a certain tolerance. If the behavior changes quickly against time, the behavior gradient is steep, and a small time step may be necessary to meet the tolerance constraint. For slow behavior, a larger time step may be possible. The varying step size along the (horizontal) time axis is depicted in Fig. 7b by horizontal arrows.

Piecewise continuous behavior interspersed with discontinuities is shown in Fig. 7c. The corresponding semantic domain is the same as the continuous behavior in Fig. 7b with the same execution domain and scheduler.

The piecewise continuous behavior with or without discontinuities may be considered to be of a hybrid dynamics nature. Indeed, the change from one piecewise segment of continuous-time behavior to the next may be invoked by the occurrence of a discrete event at the point in time of the transition.

Multidimensional Evolution Domains

In discrete-event systems such as the one illustrated in Fig. 7a, it is often desirable for two events to occur at the same point in time, either because they were scheduled as such independently or because one causes the other with no time delay.

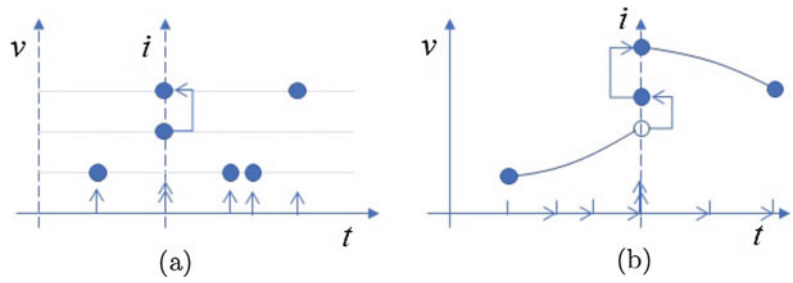
Figure 8a sketches how multiple events at a given point in time would lead to a two-dimensional evolution domain. The double-headed arrow pointing up indicates two such events. Because both of those events have the same time associated with them, in order to distinguish their moment of occurrence (e.g., to order them), an additional index is necessary. Hence, the moment of each event occurrence is represented by a couple, $\langle t, i \rangle$, that comprises the time, t, and an index, i. The index is an element from an ordered set for which the integers may be chosen (even if the metrics are irrelevant). Figure 8a indicates the additional integer dimension by a dashed axis at the time when multiple events occur. Note that many events may be dynamically scheduled at the same point in time before the next time advance.

Similar to discrete-event systems, in hybrid dynamic systems such as illustrated in Fig. 7c, multiple events may occur at the same point in time. Likewise, an additional dimension would be required in order to separate the moment of occurrence for the multiple events. Figure 8b illustrates the semantic domain with an additional axis to represent the integer part of the evolution domain at the point in time of the discontinuity where the sequence of events takes place.

Piecewise Continuous Systems

Simulation behavior that is piecewise continuous interspersed with discontinuities (e.g., the semantic domains in Figs. 7c and 8b) introduces complications that require special attention from a hybrid dynamic system perspective. Referring to Fig. 8b, four stages of execution can be recognized:

- Continuous behavior evolution that may require a solver reset in case continuity assumptions about magnitude and higher-order derivatives are violated when a discontinuity occurs.



Simulation of Hybrid Dynamic Systems, Fig. 8 Multidimensional evolution domain. (**a**) Multiple events. (**b**) Event sequence. © The MathWorks, Inc.

- Detecting when events are generated, which introduces numerical challenges around threshold crossings.
- Support for multiple mode transitions at the same point in time and handling of initial states that are inadmissible or inconsistent.
- Reinitializing state based on discontinuities that are implicit and require handling instantaneous changes as having either 0 or infinitesimal duration.

Of these, first, the numerical challenges of event detection are considered in more detail. Next, challenges in the interaction between mode transition behavior and state reinitialization are studied after which a hyperdense evolution domain is introduced. Finally, caution is drawn to a number of pathological behaviors.

Event Detection

As a behavior evolves continuously, discontinuities are specified to occur when either time or behavior values exceed certain thresholds. The former are referred to as *time events*, and their time of occurrence is known when they are scheduled. The latter are referred to as *state events* because the behavior values derive from the continuous state and the time of occurrence must be dynamically determined (i.e., as the behavior evolves) (Cellier 1979).

The threshold behavior of state events is specified by inequalities. For example, the thermostat model in Fig. 1 generates events when the temperature falls below min as specified by $x \leq min$ or when the temperature exceeds max as specified by $x \geq max$. Note that the inequalities include the threshold value (they include the equal sign) so as to have clearly defined left limit values to follow the progression of time. These causal (in a temporal sense) limits imply that the intervals of continuous evolution will be left closed and right open (Mosterman et al. 1998a).

Detecting whether the value of a variable crosses a threshold can be reformulated to detect whether a function of that variable crosses 0. This function is called an indicator function, g_α, that takes as input the continuous state of the system, x, the forcing function, u, and time, t. The indicator function may differ per discrete mode, α. The indicator function is used to determine whether a zero crossing, zc, occurred based on its sign:

$$zc_\alpha(x,u,t) = \begin{cases} 1 & if \quad g_\alpha(x,u,t) > 0 \\ 0 & if \quad g_\alpha(x,u,t) = 0 \\ -1 & if \quad g_\alpha(x,u,t) < 0 \end{cases} \quad (2)$$

In case zc changes its sign, an event is generated. For example, for the freewheeling diode model in Fig. 4b when the diode is in its *off* state ($I_D = 0$), an indicator function is

$$g_{off}(V_B, p) = V_B - R\frac{p}{L} \quad (3)$$

where p is the continuous state and V_B is the forcing function, while R and L are parameter values for the resistor and inductor, respectively. When this function changes its sign, a zero-crossing event is generated.

Detecting whether a change of sign occurs typically requires the indicator function to be evaluated positively and negatively. Thus, the indicator function fails to guard against invalid operations (e.g., taking the square root of a negative variable). Rather than returning an error, the invalid operation may return a substitute value

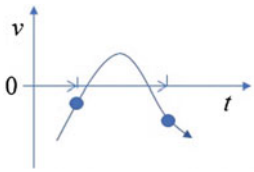
that enables successful zero-crossing detection (e.g., the square root may return a negative value based on the absolute value), while the substitute value is never considered an actual (accepted) simulation value. Note that there are techniques to prevent invalid operations but at the cost of efficiency (Esposito et al. 2001).

Even though including the equal sign in threshold comparison results in sound mathematical limit behavior, numerically it is less meaningful. It is well-known in computer programming not to check for equality involving a floating point value, for example, as a loop termination condition. Likewise, detecting whether an indicator function equals 0 is challenging. In a common approach, the 0 level is defined as a "numerical 0" by a tolerance, *tol*.

$$zc_\alpha(x,u,t) = \begin{cases} 1 & if \quad g_\alpha(x,u,t) > tol \\ 0 & if \quad -tol \le g_\alpha(x,u,t) \le tol \\ -1 & if \quad g_\alpha(x,u,t) < -tol \end{cases} \tag{4}$$

Though a numerical zero enables the determination that the value of a behavior is (numerically) 0, it complicates detecting an actual crossing of 0 (i.e., a change in sign of the indicator function). The tolerance around 0 allows a behavior to be numerically 0 for a duration of time even if the gradient is nonzero. And so the indicator function may show a sequence of signs at simulation steps as $- \rightarrow 0 \rightarrow 0 \rightarrow +$. It becomes difficult to determine whether this constitutes an actual 0 crossing or whether the behavior first settled at 0 and then started to deviate.

Another key challenge in zero-crossing detection is when an even number of crossings occur within one simulation step. Figure 9 illustrates the situation where the solid circles indicate values of the indicator function computed at two consecutive simulation steps. The dynamics of the indicator function allow its value to change from negative to positive and back without a change in sign being detected at the simulation steps. One approach to mitigating this problem is to include the indicator functions as part of the step-size control that the numerical solver uses for generating continuous-time behavior, which introduces additional computational cost.



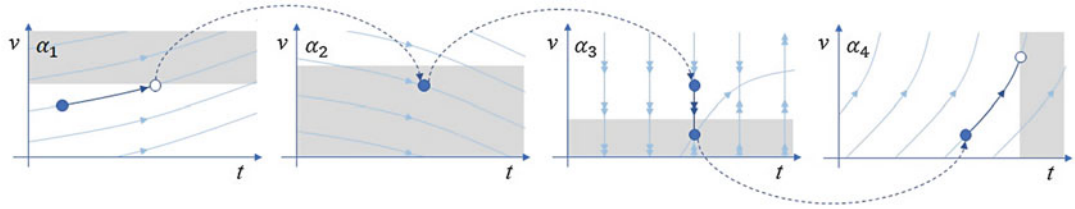
Simulation of Hybrid Dynamic Systems, Fig. 9 Zero crossing with even zeros. © The MathWorks, Inc.

After detecting a zero crossing, the point in time at which the indicator function changes sign is located within a given tolerance (different from the tolerance on the value domain). Well-understood root-finding methods such as Newton's method or a bisectional search can be used for this purpose (Park and Barton 1996; Zhang et al. 2008).

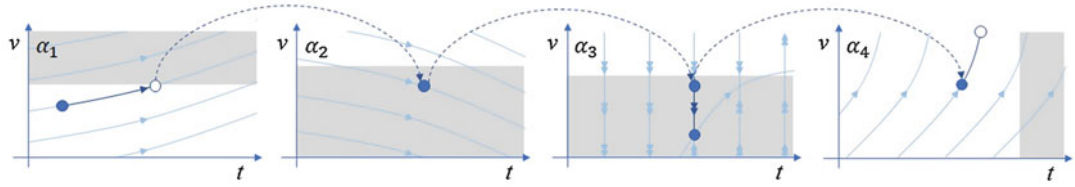
Mode Transition and State Reinitialization

The inequalities that give rise to mode transition events define where on the semantic domain behavior evolution is admitted and where not. Figure 10 depicts a sequence of mode transitions from left to right. In the initial mode, α_1, behavior evolves in continuous time from an initial state (the solid circle) according to the vector field lines.

The gray area indicates an area of the semantic domain that is inadmissible. Once the behavior value exceeds the lower bound of the area, a zero-crossing event causes a transition to mode α_2. The state of the behavior immediately prior to the mode transition taking place is used to initialize behavior in mode α_2. As shown in Fig. 10, this state is in the inadmissible area of mode α_2 causing an immediate further transition to mode α_3. Mode α_3 consists of a *jump space* as indicated by the double-headed arrows. In this space, a discontinuous change in the jump direction onto the continuous space takes place. The one-dimensional continuous space is shown as a solid curve with single-headed arrow. An example of this behavior is the freewheeling diode circuit in Fig. 4a where the flux of the inductor jumps to 0 when the switch is opened and the diode is still off. Figure 10 shows that in mode α_3 after the discontinuous jump, the state is in the inadmissible space, and a transition to mode α_4 takes place. In mode α_4, behavior evolves according to the vector field lines with



Simulation of Hybrid Dynamic Systems, Fig. 10 A sequence of mode changes. © The MathWorks, Inc.



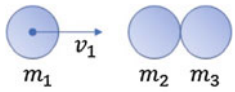
Simulation of Hybrid Dynamic Systems, Fig. 11 Alternative scenario across a sequence of mode changes. © The MathWorks, Inc.

the initial state being the state that resulted from the discontinuous jump in mode α_3. The behavior evolution in α_4 then reaches an inadmissible area based on time exceeding a threshold value, and further mode transitions may happen.

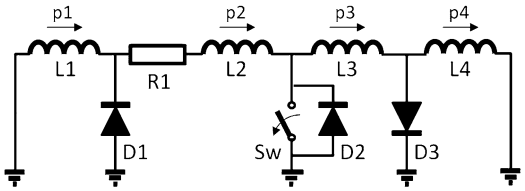
Figure 11 illustrates an alternative scenario for the sequence of mode changes in Fig. 10. In Fig. 11, the inadmissible space of α_3 is larger such that the initial state after transition from mode α_2 is in the admissible space already. Now, before the discontinuous jump in α_3 takes place, the transition to mode α_4 occurs, and behavior evolution in mode α_4 starts from a different initial value.

Note that the discontinuous change of the behavior state upon entering mode α_3 is part of the behavior specified for that mode instead of a separately specified reinitialization. This is well suited for models where the dynamics are modeled by a system of differential and algebraic equations (DAE). The algebraic part of these equations defines the jump space, whereas the differential equation part defines the continuous-time behavior evolution space. The continuous behavior evolves on a manifold in the generalized state space, while the jump space corresponds to a projection onto this manifold (Mosterman 2002; van der Schaft and Schumacher 1996; Verghese et al. 1981).

Alternatively, discontinuous change in the behavior state may be specified when a transition takes place from one mode to the next. For example, the bouncing ball model in Fig. 3 specifies a discontinuous change in velocity when a transition from one mode to the next (albeit the same) mode takes place. From a physics perspective, it may be preferred to have a mode associated with a discontinuity so the configurations of phenomena that cause the discontinuity are clear and to support compositionality. For the bouncing ball model, there is no model of the floor as a source of 0 velocity, while in general in collision models, this is preferred. Figure 12 shows Newton's cradle where in case of a perfectly elastic collision (a coefficient of restitution of 1 for the difference in velocities after and before the collision: $\Delta v^+ = -\Delta v$) with equal masses $m_1 = m_2 = m_3$, the momentum of m_1 first fully transfers to m_2 and immediately following to m_3. By modeling the masses and the collision phenomenon between the colliding masses (m_1 collides with m_2 and m_2 collides with m_3), additional masses are easily added (though great care must be taken when masses are not equal or coefficients of restitution are not 1 Mosterman (2007b)). Note that this model results in a sequence of projections, each of which requires a change in physical state



Simulation of Hybrid Dynamic Systems, Fig. 12 A series of reinitializations based on redistribution of momentum. © The MathWorks, Inc.



Simulation of Hybrid Dynamic Systems, Fig. 13 Electric circuit with different types of discontinuities. © The MathWorks, Inc.

before a new mode of continuous behavior is reached.

A Mode Transition and Reinitialization Example

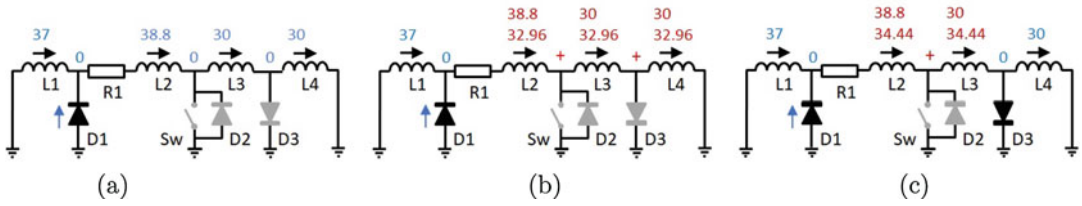
Figure 10 illustrates three classes of mode behavior:

- Continuous evolution with a state event in mode α_1 and a time event in mode α_4
- Immediate consecutive transition in mode α_2
- Transition after an instantaneous projection in mode α_3

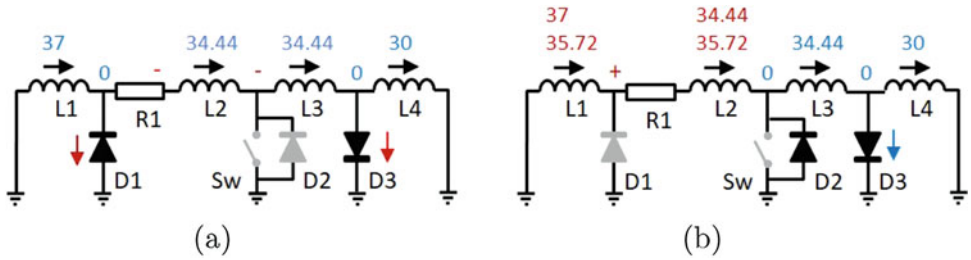
The electric circuit in Fig. 13 is an example with physical behavior that can be modeled according to each of these. The four inductors ($L1$ through $L4$) store an initial flux ($p1$ through $p4$) that corresponds to each of their currents ($I1$ through $I4$) by the equation $I = \frac{p}{L}$. The resistor, $R1$, follows Ohm's law and creates a voltage difference (positive from the left to right connection), V, based on the current through $R1$, I, as $V = I \cdot R1$. The three diodes ($D1$ through $D3$) either limit a positive voltage difference to 0 (the *on* state) or a negative current to 0 (the *off* state). Similarly, the switch, Sw, enforces a 0 voltage difference when in its *closed* state or a 0 current when in its *open* state. Initially, the switch is in its open state. Assume the inductances $L1$, $L2$, $L3$, and $L4$ to be equal to 1.

Consider the scenario that starts with a time event at t_{switch} when the switch is opened while the flux distribution is $p1 = 37$, $p2 = 38.8$, $p3 = 30$, and $p4 = 30$, as shown in Fig. 14a. With the switch closed, the voltages at each of the junctions are 0. The difference in current through $L1$ and $L2$ results in a voltage across $R1$ that is balanced by a corresponding but negative voltage across $L2$. The open switch results in an inconsistent physical state (flux distribution) as opening the branch forces $p2$, $p3$, and $p4$ to be equal (given that $L2 = L3 = L4$). The increase in $p4$ induces a voltage spike as indicated by the + sign in Fig. 14b which causes diode $D3$ to come on and force the junction to a 0 voltage instead as illustrated in Fig. 14c. Because $D3$ comes on as a result of the projected change in flux distribution in Fig. 14b, that flux distribution is never achieved, and redistribution in Fig. 14c is determined based on the original flux distribution in Fig. 14a. Since the intermediate mode in Fig. 14b does not affect the ultimate physical state, it is called a *mythical mode* (Mosterman et al. 1998b; Nishida and Doshita 1987).

Once a consistent flux distribution is determined, it can be adopted as the current state. Because physical in nature, redistributing fluxes is modeled to require an infinitesimal period of time, ϵ, and the event iteration resumes at this new point in time, $t_{\text{switch}} + \epsilon$. Figure 15a shows the adopted consistent state and the voltages and currents that correspond to the new state. Because $p1$ has become larger than $p2$, a negative current would flow through $D1$ and so the diode turns off. Also, the positive current flow through $R1$ that is induced by $p2$ causes a positive voltage drop. Because the junction to the left of $R1$ is at 0 volt, the right-hand port of $R1$ is negative. With $L2 = L3$ and $D2$ off, the voltage drop across both $L2$ and $L3$ must be equal, which is half the positive voltage drop across $R1$. This negative voltage causes $D2$ to come on. Figure 15b shows the new mode with the newly required flux distribution because $L1$ and $L2$ are now coupled, while $L3$ has become decoupled.



Simulation of Hybrid Dynamic Systems, Fig. 14 When the switch opens, a new consistent flux distribution is determined. (**a**) $\langle t_{\text{switch}}, 0\rangle$. (**b**) $\langle t_{\text{switch}}, 1\rangle$. (**c**) $\langle t_{\text{switch}}, 2\rangle$. © The MathWorks, Inc.



Simulation of Hybrid Dynamic Systems, Fig. 15 Time is advanced by ϵ to model the physical state (flux) change and a consecutive change follows. (**a**) $\langle t_{\text{switch}} + \epsilon, 0\rangle$. (**b**) $\langle t_{\text{switch}} + \epsilon, 1\rangle$. © The MathWorks, Inc.

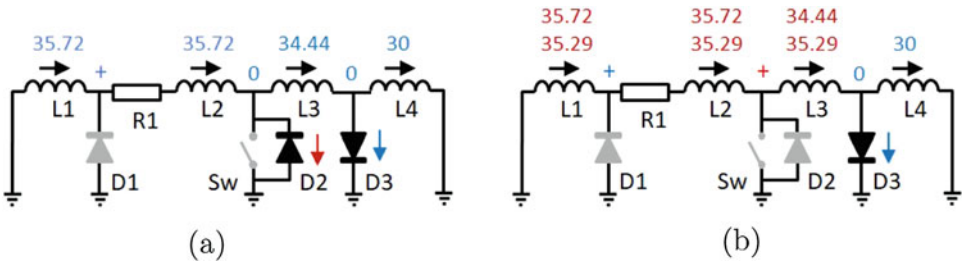
Since no further mode changes occur, time is advanced by an infinitesimal amount, and the flux redistribution is adopted as the next physical state. Figure 16a shows this mode and flux distribution as the starting point for the next event iteration. Because the changed flux of $L2$ exceeds that of $L3$, $D2$ would have a negative current and so $D2$ turns off. As a result, $L3$ is coupled with $L1$ and $L2$, and Fig. 16b shows the corresponding flux redistribution. At this point, no further mode changes occur, time is advanced by an infinitesimal amount, and the redistributed fluxes are adopted.

Figure 17a shows the flux distribution at $t_{\text{switch}} + 3\epsilon$ as well as qualitative values for the voltages and currents. The current through $R1$ causes a positive voltage drop that is equally distributed as negative voltage across each of the inductors $L1$, $L2$, and $L3$ such that the sum of voltages equals 0. This causes $D2$ to turn on, as shown in Fig. 17b. At this point, no further mode changes occur and no flux redistribution is required. Therefore, the sequence of mode changes, and reinitialization terminates so that continuous-time behavior evolution resumes.

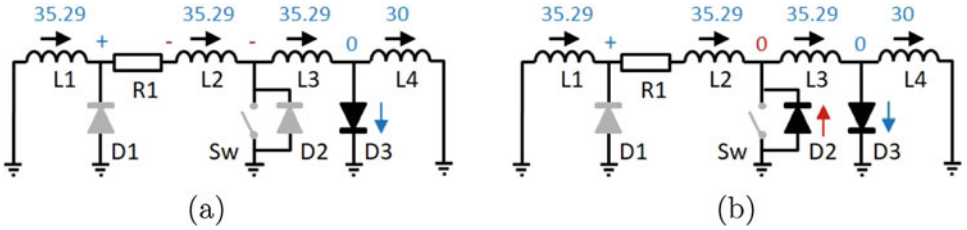
Modeling Diode Behavior

Figure 18 shows the diode model used for simulation of the electrical circuit in Fig. 13. Note that the switching conditions do not include 0 (i.e., the voltage must exceed 0 or the current must fall below 0 before being limited to 0). Otherwise, consider the case where a diode is in its *on* state and enforcing a 0 voltage when it is determined that the current would fall below 0. In this situation, the diode would change its state to *off*, but without any changes in the physical state of the circuit, the corresponding voltage would still be 0. Hence, if the switching conditions would include 0, the diode would immediately switch back to its *off* state, and an infinite loop of state changes from *on* to *off* and back arises.

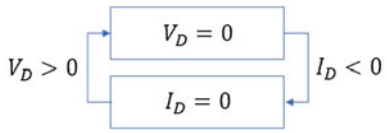
Mathematically, not including the boundary of a switching area may be complicating since the point in time of switching and the limit value at this point are not well defined. A threshold value that must be exceeded eliminates this concern (e.g., switching conditions $I_D \leq I_{th}$ instead of $I_D \leq 0$ and $V_D \geq V_{th}$ instead of $V_D \geq 0$ in Fig. 4b). In simulation, however, the threshold values are a numerical effect when the switching



Simulation of Hybrid Dynamic Systems, Fig. 16 Time is advanced by another ϵ to model the physical state (flux) change to introduce a "stutter". (**a**) $\langle t_{\text{switch}} + 2\epsilon, 0 \rangle$. (**b**) $\langle t_{\text{switch}} + 2\epsilon, 1 \rangle$. © The MathWorks, Inc.



Simulation of Hybrid Dynamic Systems, Fig. 17 A consistent physical state leads to another mode transition with the same consistent physical state. (**a**) $\langle t_{\text{switch}} + 3\epsilon, 0 \rangle$. (**b**) $\langle t_{\text{switch}} + 3\epsilon, 1 \rangle$. © The MathWorks, Inc.



Simulation of Hybrid Dynamic Systems, Fig. 18 Finite state machine model of a diode. © The MathWorks, Inc.

conditions $I_D < 0$ and $V_D > 0$ are used and a threshold value is not strictly necessary.

Also note that the diode model in Fig. 18 is formulated as a finite state machine, whereas sequential logic is not necessary to capture the desired behavior. For example, in a complementarity formulation (van der Schaft and Schumacher 1996), an ideal diode is modeled by an equality that requires either I_D or V_D to be 0 while complementing the equation with inequalities that require either the voltage to be less than or equal to 0 or the current to be larger than or equal to 0:

$$\begin{aligned} 0 &= V_D I_D \\ -V_D &\geq 0 \\ I_D &\geq 0 \end{aligned} \tag{5}$$

Complementarity formulations are effective in modeling discontinuous behavior of different forms (e.g., rigid body mechanics (Pfeiffer and Glocker 1996)). The use of sequential logic covers a broader range of phenomena in hybrid dynamic systems, though, and sometimes is required (e.g., a latched bitank system (Mosterman and Biswas 1995)).

A Hyperdense Semantic Domain and Numerical Simulation

As the example in Fig. 13 shows, the semantic domain of piecewise continuous models of physical systems requires three aspects:

- A continuous domain for continuous-time behavior evolution
- An infinitesimal domain for discontinuous changes in physical state
- An ordered domain for mode changes

The ability to separate discontinuous change in physical state from mode changes is illustrated in Fig. 14c. With the flux distribution at t_{switch}, the current through $D1$ keeps $D1$ on. The current at $t_{\text{switch}} + \epsilon$, however, is such that $D1$ turns

off. If the infinitesimal time step would not be required to advance the flux distribution, $D1$ would turn off before $D2$ turns on, and a different behavior from what is shown in Fig. 15b would be generated.

The set of hyperreals, $*\mathbb{R}$, of nonstandard analysis (Keisler 2012) embeds infinitesimals in the set of reals, $\mathbb{R}$, and so $*\mathbb{R}$ supports the semantic domain for continuous-time behavior evolution and infinitesimal time steps. In addition, the set of integers support the ordered domain of mode changes. Hence, $*\mathbb{R} \times \mathbb{N}$ provides a *hyperdense* semantic domain that supports the necessary detail to separately evaluate the discontinuous and mode change behavior (Mosterman et al. 2014).

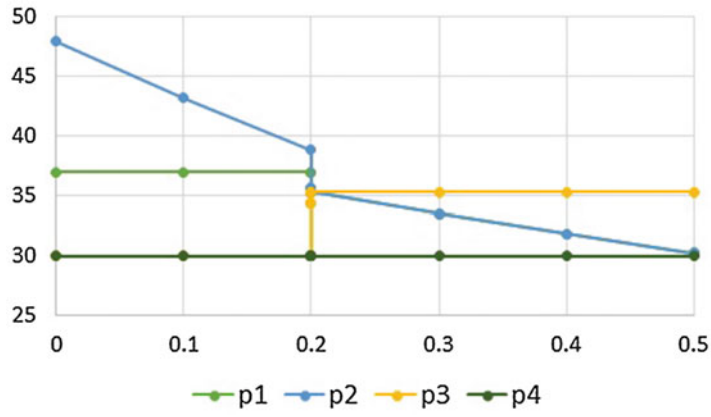
Numerical simulation requires discrete-time steps to evolve behavior on the reals, $\mathbb{R}$, while the infinitesimal time advances can be captured by an integer count of infinitesimal steps during mode transition sequences. The resulting domain for numerical simulation becomes three dimensional as $\mathbb{R} \times \mathbb{N} \times \mathbb{N}$. For example, the mathematical moment of evaluation in Fig. 14a is $\langle t_{\text{switch}}, 0\rangle$ with the numerical simulation step being identified as $\langle t_{\text{switch}}, 0, 0\rangle$. Likewise, the mathematical moment of evaluation in Fig. 17b is $\langle t_{\text{switch}} + 3\epsilon, 1\rangle$ with the numerical simulation step being identified as $\langle t_{\text{switch}}, 3, 1\rangle$.

The results of a simulation with HyBrSim (Mosterman 2002) where $R1 = L1 = L2 = L3 = L4 = 1$ from time 0 to 0.5s are shown in Fig. 19. Before the switch is opened at $t = t_{\text{switch}} = 0.2s$, the flux $p2$ decreases because of the dissipative effect of $R1$. After the switch is opened and the mode transition sequence has converged, the fluxes $p1$ and $p2$ decrease equally (overlapping traces in the graph) because $D1$ is *off*. Figure 20 shows the mode transition sequence at $t = 0.2s$ in detail where the indices represent the evolution tuples as $1 : \langle 0.2, 0, 0\rangle$, $2 : \langle 0.2, \epsilon, 0\rangle$, $3 : \langle 0.2, 2\epsilon, 0\rangle$, $4 : \langle 0.2, 3\epsilon, 0\rangle$, and finally $5 : \langle 0.2, 3\epsilon, 1\rangle$. Figure 20a shows the consistent modes for each point in time including the initial mode when the switch is still closed. The overall mode is comprised of a combination of the discrete state of three junctions where the first and last junction are fully determined by the state of the corresponding diodes, $D1$ and $D3$, respectively. The state of the second junction depends on the combination of the switch, Sw, and diode, $D2$. If the switch is closed or the diode is *on*, the junction is *on*, whereas the junction is *off* otherwise. Figure 20b shows the consistent flux distributions for each of the modes in Fig. 20a with the initial distribution of the fluxes immediately before the switch is opened (t_{switch}). Note that the intermediate modes (before an accepted flux distribution is reached) are not logged in simulation; only values in time as simulated hyperreals (i.e., a tuple of time and number of infinitesimal steps from a point in real time) are shown.

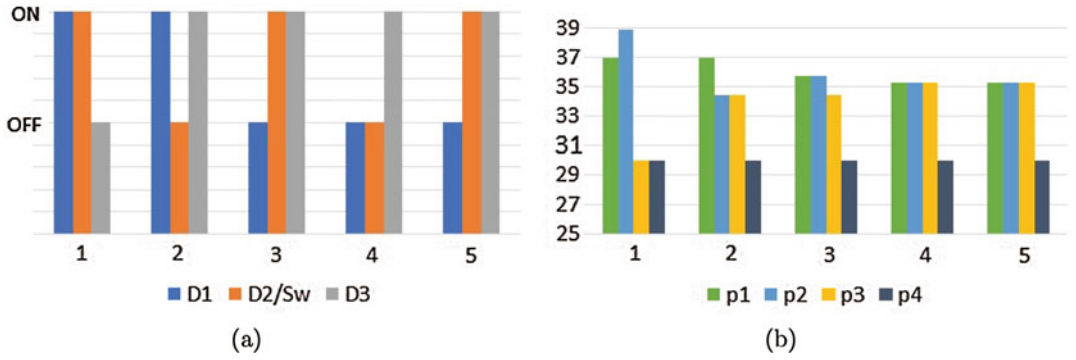
Pathological Behavior

Simulation of mode transition behavior may encounter a number of behaviors that are challenging to handle. First, the hybrid dynamic system may exhibit sliding behavior as illustrated in Fig. 21. When the behavior reaches the switching (inadmissible) area (where the boundary is included in the switching condition) in mode α_1, a transition is made to mode α_2. The behavior in mode α_2 starts on the boundary of the switching (inadmissible) area where the boundary is not included. As soon as continuous-time simulation restarts, the mode transition back to α_1 occurs. After a similarly small time step, the dynamic field lines push the behavior in mode α_1 back to the switching boundary, and repeated switching occurs with the minimum step size that the numerical simulation allows. Such behavior is, for example, the purview of *sliding mode control* (such as used in antilock braking systems), and special simulation algorithms (Mosterman et al. 1999) can be used to filter out the fast switching behavior while retaining the slow dynamic behavior along the switching boundary.

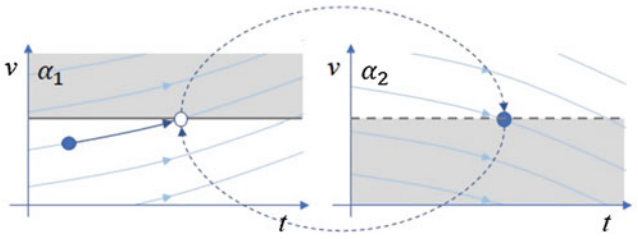
Second, in case of overlapping switching areas in Fig. 21, as soon as the switching boundary in mode α_1 is reached, the behavior enters an infinite loop of mode transitions that occur in 0 time. Hence, simulation of the dynamic behavior halts, which is typically undesirable (e.g., because this is inconsistent with observed behavior in the physical world). This behavior may be difficult to



Simulation of Hybrid Dynamic Systems, Fig. 19 Simulation of flux distribution over time. © The MathWorks, Inc.



Simulation of Hybrid Dynamic Systems, Fig. 20 Simulation results of the mode transition sequence. (**a**) Mode transitions. (**b**) Flux redistributions. © The MathWorks, Inc.



Simulation of Hybrid Dynamic Systems, Fig. 21 Sliding mode behavior. © The MathWorks, Inc.

determine as present in a hybrid dynamic system model and often cannot be detected other than when exhibited during simulation.

Third, mode transitions may occur at progressively smaller time steps such as described for the bouncing ball model in Fig. 3. In an ideal mathematical model, time converges to a limit point. In a numerical simulation, behavior may be ill defined when the numerical time step moves past the limit point. Note that the existence of a limit point may be difficult to detect, even during simulation.

Multiparadigm Modeling

Modern engineered systems increasingly rely on computational elements (e.g., microprocessors, microcontrollers, and field-programmable gate arrays) and a communication infrastructure (e.g., wired or wireless networks). Even if such engineered systems are not intrinsically hybrid dynamic systems, the physics are often best represented at a macroscopic level by

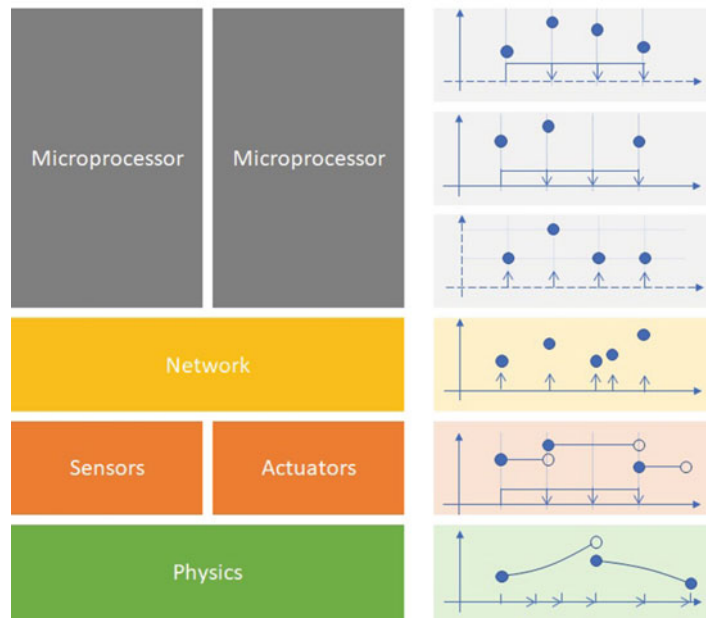
models with continuous-time semantics, and the computation is often best represented by models with discrete-time or discrete-event semantics. A structure of such systems with the various different components is shown in Fig. 22. At the bottom is the physical world. Sensors and actuators connect the physical world with the digital world via built-in computational elements. The network connectivity of these sensors and actuators enables communication with other sensors and actuators as well as powerful microprocessors or other computational elements.

Different formalisms are typically required to model the behavior of the various different components. At the microprocessor level, program code (semantic domain of Fig. 5a), synchronous data flow and difference equations (semantic domain of Fig. 5b), as well as state transition diagrams (semantic domain of Fig. 6b) are often used. Modeling network behavior generally requires the ability to capture different event densities over time for which discrete-event systems are preferred (semantic domain of Fig. 7a) (Cassandras et al. 2006; Cervin et al. 2003; Zeigler et al. 2000). Because of the interaction with the continuous-time models of the physical world, sensors and actuators are often modeled as discrete time with zero-order hold (semantic domain of Fig. 5c). Finally, the macroscopic behavior of physical systems is well represented by differential equations with discontinuous changes that occur at specific points in time (semantic domain of Fig. 7c).

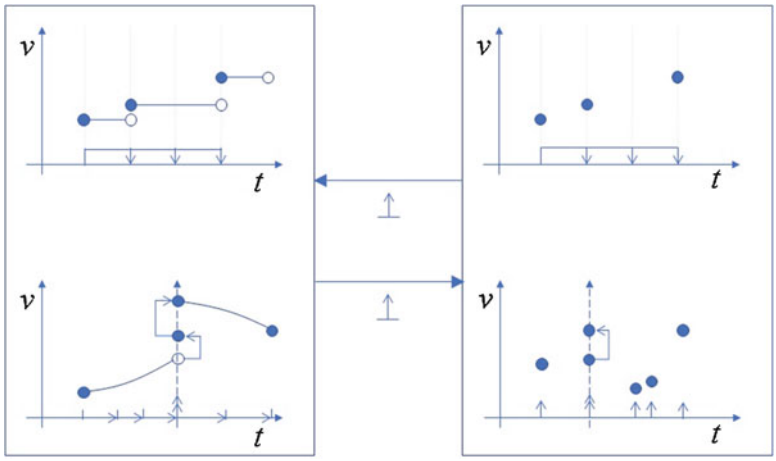
The combination of this scala of modeling formalisms with the variety of semantic domains to be used for simulation is the purview of multi-paradigm modeling (Mosterman and Vangheluwe 2004). Hybrid dynamic systems are thus fundamental to multiparadigm modeling where hybrid means combining different semantic domains.

Behavior Generation Engines

In terms of efficient execution, handling continuous time is the first and foremost challenge. Continuous-time behavior progresses along the gradient of differential equations. When thresholds are exceeded, zero-crossing events are detected and then located, and their effect is evaluated before time progresses further. Because of the progression of state values along time, this is called time-driven execution. In contrast,



Simulation of Hybrid Dynamic Systems, Fig. 22 Multiple semantic domains in engineered system modeling. © The MathWorks, Inc.



Simulation of Hybrid Dynamic Systems, Fig. 23 Time-driven and event-driven execution engines. © The MathWorks, Inc.

discrete-event behavior schedules events in the future by keeping an event calendar. The event that is scheduled earliest in time is removed from the calendar, current time is set to the event time, and the event is processed. This processing may cause further events to be scheduled or already scheduled events to be canceled (removed from the calendar). Once processing completes, the current time is set to the time of the earliest scheduled event, and the event is processed. Because time evolves in discrete steps based on scheduled events, this is called an event-driven execution (Mosterman 2007a).

Figure 23 depicts a time-driven execution engine on the left-hand side and an event-driven execution engine on the right-hand side. The bottom traces represent continuous-time evolution on the left and discrete-event evolution on the right. Note that at event times (e.g., zero-crossing events or scheduled discrete events), multiple evaluation steps may take place that may be scheduled as events in turn. Also, in case of piecewise continuous behavior, a series of infinitesimal steps may be scheduled before time-driven execution resumes.

Statically scheduled events because of periodic behavior (e.g., discrete time with a single or multiple rates and synchronous data flow) can either be included in a time-driven or an event-driven execution engine. Figure 23 shows discrete-time behavior as part of the time-driven execution engine because the zero-order hold behavior closely aligns with the continuous-time behavior and is typically used because of interacting behavior. Without hold behavior, complicated semantic challenges may arise in interacting with minor time steps of numerical solvers (Denckla and Mosterman 2008). Otherwise, the discrete-time behavior schedules a time event, and the numerical solver will evaluate the differential equations at that point in time, then evaluate the discrete-time operations, and resume continuous-time behavior.

Figure 23 shows synchronous behavior as part of the event-driven execution engine because of the discretized time domain. Also, data flow comes in statically scheduled and dynamically scheduled forms and hence is more closely related to discrete-event systems.

While the architecture in Fig. 23 allows the most efficient simulation mechanism (time driven vs. event driven) for each evolution domain, the moments of interaction must be handled efficiently as well. In one form, the interaction may consist of the time-driven model part scheduling or canceling an event for the event-driven model part (e.g., when the strength of a communicating signal falls below a threshold level, a message may be sent to a transmitter). In another form, the event-driven model part may schedule an event for the time-driven model part (e.g., a network message may set a threshold value for a valve to open). As a result, when an interacting event is scheduled or canceled, the behavior evolution of the receiving model part may change.

Two basic approaches can be employed to handle the interaction effects:

- With *conservative execution*, the engines operate in lock step. The event-driven execution engine may change its current time to be ahead of the current time of the time-driven execution engine but first stores the full state of the event-driven execution engine. The time-driven execution engine then catches up before the event-driven engine makes another step and the process repeats. If the time-driven model part generates an interacting event while in the process of catching up, the event-driven execution engine restores its full state, the interacting event is added to the event calendar, and the interacting event is processed next.
- With *optimistic execution*, the engines are allowed to process more than one step ahead of each other. For example, it may be efficient both for the event-driven engine and the time-driven engine if one processes a number of events before the other catches up. In this scheme, it becomes key to determine when the full engine state should be stored to enable quick reconstruction of the state at which an interacting event occurred. In advanced methods, an intermediate state may be reconstructed in approximation by an interpolation polynomial (which may be more efficient than reconstructing a state via simulation).

Conclusions

Hybrid dynamic systems may refer to various combinations of behavior and execution semantics:

- A continuous evolution domain with a combination of continuous and discrete behavior in the value domain.
- The cross product of evolution domains such as a continuous domain combined with an integer domain.
- A combination of different semantic domains such as discrete-time and continuous-time behavior.
- A combination of different execution engines, for example, an event-driven engine combined with a time-driven engine.

Generating an event from a continuous-time behavior via zero-crossing detection and location is both challenging to be sound and to be efficient. Generating a new mode of continuous-time behavior with a consistent continuous-time state from discrete changes is challenging as well. In particular, among the richest of hybrid dynamic system behaviors is the piecewise continuous semantic domain. With a continuous value domain, a three-dimensional hyperdense evolution domain emerges to account for sequences of state reinitialization and mode transition behavior.

Note that modeling mode transitions as either occurring in 0 time or in infinitesimal time has been shown to be an important consideration beyond simulation, namely, in reasoning about discontinuous change, with corresponding direct and approximate reasoning methods, respectively (Nishida and Doshita 1987).

Much progress has been made in hybrid dynamic system simulation. Still, open challenges remain. These include how to:

- Solve or resolve some of the pathological behaviors.
  - Statically (i.e., before simulation) determine whether the simulation may reach an infinite loop of 0 time mode transitions.
  - Find the limit points for converging time if present.
  - Develop robust simulation algorithms for higher-dimensional sliding behavior.
- Simulate impulsive behavior (e.g., support comparison of impulsive quantities such as voltage spikes and impulsive forces).

As a caution in closing, combining continuous behavior with discrete changes results in behavior that is highly sensitive to perturbations in the

initial state and parameters of the model but also to the solver parameters. Without a fundamental solution, good practice is to check sensitivity effects by using different solvers and parameters.

Cross-References

- ▸ Control Hierarchy of Large Processing Plants: An Overview
- ▸ Discrete Event Systems and Hybrid Systems, Connections Between
- ▸ Hybrid Dynamical Systems, Feedback Control of
- ▸ Modeling Hybrid Systems
- ▸ Multi-domain Modeling and Simulation
- ▸ Sliding Mode Control: Robust Finite-Time-Exact Observation and Regulation
- ▸ Switching Adaptive Control

Bibliography

Alur R, Dill DL (1994) A theory of timed automata. Theor Comput Sci 126:183–235

Benveniste A, Caspi P, Edwards SA, Halbwachs N, Guernic PL, de Simone R (2003) The synchronous languages twelve years later. Proc IEEE 91(1):64–83

Breedveld PC (1996) The context-dependent trade-off between conceptual and computational complexity illustrated by the modeling and simulation of colliding objects. In: CESA '96 IMACS Multiconference, Ecole Centrale de Lille, Lille

Cassandras CG, Clune MI, Mosterman PJ (2006) Hybrid system simulation with simevents. In: Cassandras C, Giua A, Seatzu C, Zaytoon J (eds) Analysis and design of hybrid systems 2006. Elsevier, Amsterdam, pp 267–269

Cellier FE (1979) Combined continuous/discrete system simulation by use of digital computers: techniques and tools. Ph.D. dissertation, ETH, Zurich

Cellier FE, Kofman E (2006) Continuous System Simulation. Springer, Berlin, Heidelberg

Cervin A, Henriksson D, Lincoln B, Eker J, Årzén KE (2003) How does control timing affect performance? analysis and simulation of timing using jitterbug and truetime. IEEE Control Syst Mag 23(3):16–30

Denckla B, Mosterman PJ (2008) Stream- and state-based semantics of hierarchy in block diagrams. In: Proceedings of the 17th IFAC World Congress, Seoul, pp 7955–7960

Esposito JM, Kumar V, Pappas GJ (2001) Accurate event detection for simulating hybrid systems. In: Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control, HSCC '01. Springer, London, UK, pp 204–217

Keisler HJ (2012) Elementary calculus: an infinitesimal approach, 3rd edn. Prindle, Weber and Schmidt, Dover, Mineola

MathWorks® (2019) MATLAB® User's Guide

Mosterman PJ (2002) HyBrSim—a modeling and simulation environment for hybrid bond graphs. J Syst Control Eng 216(1):35–46

Mosterman PJ (2007a) Hybrid dynamic systems: modeling and execution. In: Fishwick PA (ed) Handbook of dynamic system modeling. CRC Press, Boca Raton, chap 15, pp 15-1–15-26

Mosterman PJ (2007b) On the normal component of centralized frictionless collision sequences. ASME J Appl Mech 74(5):908–915

Mosterman PJ, Biswas G (1995) Behavior generation using model switching: a hybrid bond graph modeling technique. In: Cellier FE, Granda JJ (eds) 1995 International conference on bond graph modeling and simulation (ICBGM '95). Society for Computer Simulation, Simulation Councils, Inc., Las Vegas, no. 1 in Simulation, vol 27, pp 177–182

Mosterman PJ, Vangheluwe H (2004) Computer automated multi-paradigm modeling: an introduction. Simul Trans Soc Model Simul 80:433–450

Mosterman PJ, Biswas G, Sztipanovits J (1998a) A hybrid modeling and verification paradigm for embedded control systems. Control Eng Pract 6:511–521

Mosterman PJ, Zhao F, Biswas G (1998b) An ontology for transitions in physical dynamic systems. In: AAAI98, pp 219–224

Mosterman PJ, Zhao F, Biswas G (1999) Sliding mode model semantics and simulation for hybrid systems. In: Antsaklis P, Kohn W, Lemmon M, Nerode A, Sastry S (eds) Hybrid systems V. Lecture notes in computer science. Springer, Berlin/Heidelberg, pp 218–237

Mosterman PJ, Simko G, Zander J, Han Z (2014) A hyperdense semantic domain for hybrid dynamic systems to model different classes of discontinuities. In: Proceedings of the 17th international conference on hybrid systems: computation and control, HSCC '14. ACM, New York, pp 83–92

Murata T (1989) Petri nets: Properties, analysis and applications. Proc IEEE 77(4):541–580

Nishida T, Doshita S (1987) Reasoning about discontinuous change. In: Proceedings AAAI-87, Seattle, Washington, pp 643–648

Park T, Barton PI (1996) State event location in differential-algebraic models. ACM Trans Model Comput Simul 6(2):137–165

Pfeiffer F, Glocker C (1996) Multibody dynamics with unilateral contacts. John Wiley & Sons, Inc., New York

van der Schaft AJ, Schumacher JM (1996) The complementary-slackness of hybrid systems. Math Control Signals Syst 9:266–301

Verghese GC, Lévy BC, Kailath T (1981) A generalized state-space for singular systems. IEEE Trans Autom Control 26(4):811–831

Zeigler BP, Kim TG, Praehofer H (2000) Theory of modeling and simulation, 2nd edn. Academic Press, Inc., Orlando

Zhang F, Yeddanapudi M, Mosterman PJ (2008) Zero-crossing location and detection algorithms for hybrid system simulation. In: Proceedings of the 17th IFAC world congress, Seoul, pp 7967–7972