Introduction to Hybrid Systems

Michael S. Branicky

Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, OH 44106, U.S.A.
mb@case.edu

Summary. Hybrid systems arise when the continuous and the discrete meet. Combine continuous and discrete inputs, outputs, states, or dynamics, and you have a hybrid system. Particularly, hybrid systems arise from the use of finite-state logic to govern continuous physical processes (as in embedded control systems) or from topological and network constraints interacting with continuous control (as in networked control systems). This chapter provides an introduction to hybrid systems, building them up first from the completely continuous side and then from the completely discrete side. It should be accessible to control theorists and computer scientists alike.

1 Hybrid Systems All Around Us

Hybrid systems arise in embedded control when digital controllers, computers, and subsystems modeled as finite-state machines are coupled with controllers and plants modeled by partial or ordinary differential equations or difference equations. Thus, such systems arise whenever one mixes logical decision making with the generation of continuous-valued control laws. These systems are driven on our streets, used in our factories, and flown in our skies; see Fig. 1.

Adding to the complexity is the case where sensing, control, and actuation are not hardwired but connected by a shared network medium; see Fig. 2. Such *networked control systems* (*NCSs*) are an important class of hybrid control systems [40]. The hybrid nature inherent in embedded control is further complicated by (i) the asynchronous or *event-driven* nature of data transmission, due to sampling schemes, varying transmission delay, and packet loss; and (ii) the discrete implementation of the network and its protocols, involving packets of data, queuing, routing, scheduling, etc.

So, hybrid systems arise in embedded and networked control. More specifically, real-world examples of hybrid systems include systems with relays, switches, and hysteresis [33,37]; computer disk drives [18]; transmissions, stepper motors, and other motion controllers [14]; constrained robotic systems [4];

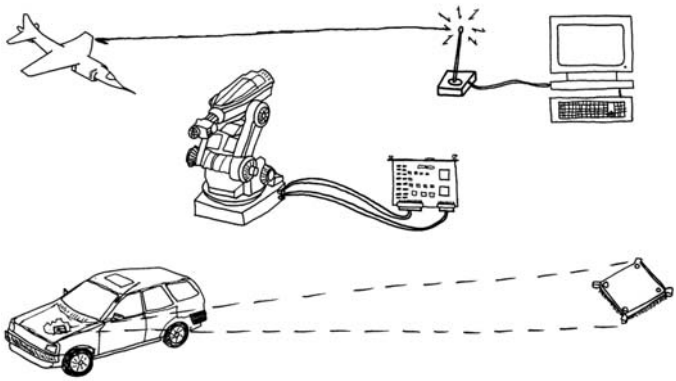

Fig. 1. Hybrid systems arising from embedded control

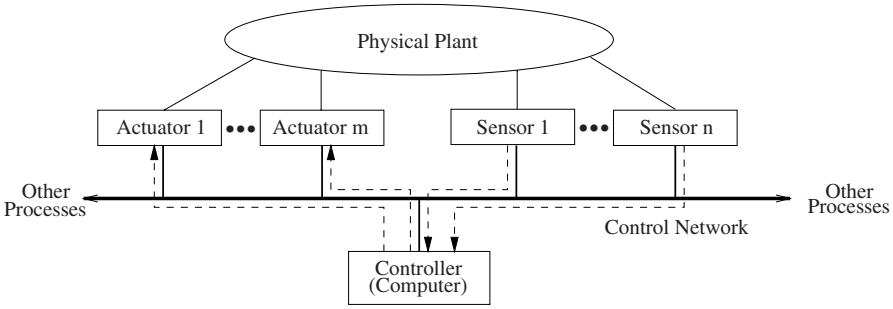

Fig. 2. Typical networked control system setup and information flows [40]

automated highway systems (AHSs) [36]; flight control and management systems [27, 35]; multi-vehicle formations and coordination [32]; analog/digital circuit codesign and verification [26]; and biological applications [17]. Next, we examine some of these in more detail.

Systems with switches and relays: Physical systems with switches and relays can naturally be modeled as hybrid systems. Sometimes, the dynamics may be considered merely discontinuous, such as in a blown fuse. In many cases of interest, however, the switching mechanism has some hysteresis, yielding a discrete state on which the dynamics depends. This situation is depicted by the multi-valued function H shown in Fig. 3(left). Suppose that the function H models the hysteretic behavior of a thermostat. Then a thermostatically controlled room may be modeled as follows:

$$\dot{x} = f(x, H(x - x_0), u), \tag{1}$$

where x and x_0 denote actual and desired room temperature, respectively. The function f denotes the dynamics of temperature, which depends on the current temperature, whether the furnace is switched On or Off, and some

auxiliary control signal u (e.g., the fuel burn rate). Note that this system is not just a differential equation whose right-hand side is piecewise continuous. There is "memory" in the system, which affects the value of the vector field. Indeed, such a system naturally has a finite automaton associated with the hysteresis function H, as pictured in Fig. 3(right). The notation ![condition] denotes that the transition *must* be taken when "enabled." That is, the event of x attaining a value greater than or equal to Δ triggers the discrete or *phase* transition of the underlying automaton from $+1$ to -1.

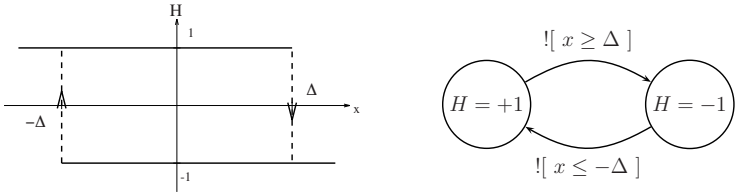

Fig. 3. (left) Hysteresis function, H, (right) finite automaton associated with H

Disk drive: Once a computer disk drive is up and spinning, it receives external commands to find data. The action of the disk drive is modeled by the differential (or difference equations) capturing the dynamic behavior of the disk, spindle, disk arm, and motors. The drive receives symbolic inputs of disk sectors and locations, it waits until the head is settled on the appropriate cylinder, begins a read operation and then transmits symbolic outputs corresponding to the bytes read (see Fig. 4). Again, the edge labels ![condition] denote transitions taken as soon as the condition is true; Read and Seek(Adr) are symbolic commands issued from another element/process in the system. While the logic governing this cycle of operation is simple, the vast majority of logic inside a disk drive (not shown, see [18]) is for startup, shutdown, and error handling.

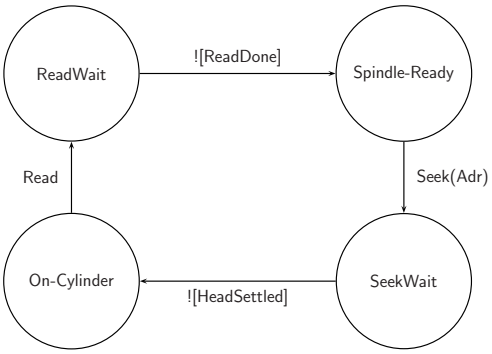

Fig. 4. Hybrid system associated with main disk drive functionality

Hopping robot control: Constrained robotic systems are interesting examples of hybrid systems. In particular, consider the hopping robots of Marc Raibert [29]. The dynamics of these devices are governed by gravity, as well as the forces generated by passive and active (pneumatic) springs. The dynamics change abruptly at certain event times and fall into distinct phases: Flight, Compression, Thrust, and Decompression; see Fig. 5(left). In fact, Raibert has built controllers for these machines that embed a finite-state machine that transitions according to these detected phases; see Fig. 5(right). There, the variable h is the distance of the hopper's base from the ground, and x is the displacement of the spring from equilibrium; T is an auxiliary timer, and the notation /action denotes that on transitioning, T is reset to zero; $\wedge$ denotes logical "and." Therefore, the transition from Flight to Compression occurs when touchdown is detected; that from Decompression to Flight upon liftoff. The switch from Compression to Thrust is controller initiated: when it detects the bottom-most point of compression, it activates the pneumatic cylinders to open for a fixed period of time, τ_{thrust}, after which Decompression of the pneumatic spring continues. Thus, finite automata and differential equations naturally interact in such devices and their controllers.

Vehicle powertrains: An automobile transmission system takes the continuous inputs of accelerator position and engine RPM and the discrete input of gear position and translates them into the motion of the vehicle. Likewise, the engine has discrete cylinders, which are fired at certain event times, as well as the continuous variables of fuel/air mixture and temperature. Finally, the whole powertrain is governed by a number of networked microprocessors to achieve some overall goal. For example, they may be coordinated by a cruise control system that accelerates and decelerates under different profiles. The desired profile is chosen depending on sensor readings (e.g., continuous reading of elevation, discrete coding of road condition, etc.). In such a case, we are to design a control system with both continuous and discrete inputs and outputs, whose internal components themselves are hybrid. This is typical in hybrid control; see Fig. 6. There, I and O are discrete (i.e., countable) sets of symbols and U and Y are continuums.

AHS: A more complicated example of a hybrid system arises in the control structures for an automated highway system (AHS) [36]. The basic goal of one such system is to increase highway throughput by means of a technique known as *platooning*. A platoon is a group of between, say, one and twenty vehicles traveling closely together in a highway lane at high speeds. To ensure safety—and proper formation and dissolution of structured platoons from the "free agents" of single vehicles—requires a bit of control effort! Protocols for basic maneuvers such as Merge, Split, and ChangeLane have been proposed in terms of finite-state machines. More conventional controllers govern the engines and brakes of individual vehicles. Clearly, the system is hybrid, with each vehicle

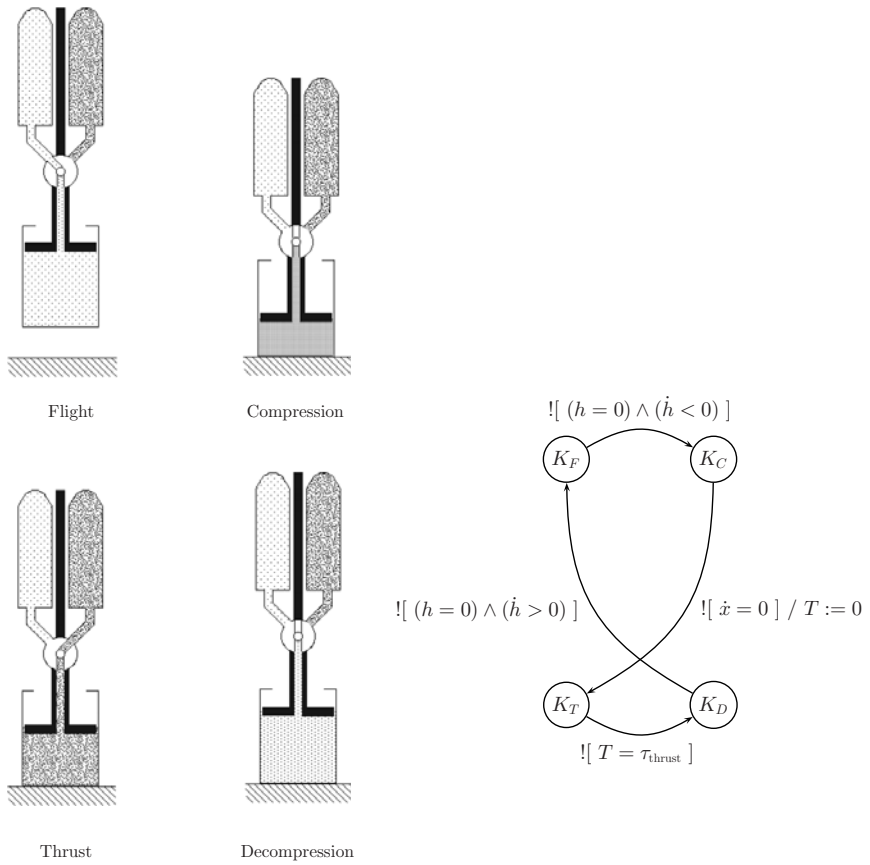

Fig. 5. Raibert's hopping robot: (left) dynamic phases (reproduced from [4], p. 260, Fig. 3, © Springer-Verlag), (right) finite-state controller with transitions specified

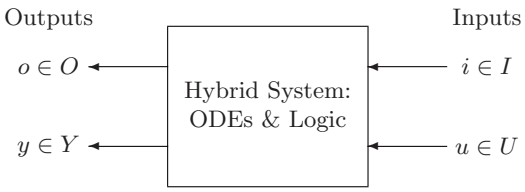

Fig. 6. Block diagram of prototypical hybrid control system

having a state determined by continuous variables (such as velocity, engine RPM, and distance to car ahead) and the finite states of its protocol-enacting state machines.

Flight control: Flight controllers are organized around the idea of *modes*. For example, one might easily imagine different control schemes for Take-Off, Ascend, Cruise, Descend, and Land. More complex is a whole flight vehicle management system, which coordinates flight in these different regimes, while also planning flight paths considering air traffic, weather, fuel economy, and passenger comfort [27].

Preview

Now that we have given a taste of the hybrid nature of real applications, we turn to developing mathematical models that can capture their behavior. Instead of proceeding directly to such hybrid models, we build them up, step by step, from well-known models of completely continuous (viz., ODEs in Section 2) and completely discrete (viz., finite automata in Section 3) systems. Then, in Section 4, we present an overarching model of hybrid systems that combines ODEs and automata. Throughout, we fix ideas using examples.

2 From Continuous Toward Hybrid

In this section, we review ordinary differential equations (ODEs) as a base continuous model,[1] then show how to add various discrete phenomena to them, as seen in the applications above.

2.1 Base continuous model: ODEs

In this chapter, the base continuous dynamical systems dealt with are defined by the solutions of *ODEs*:

$$\dot{x}(t) = f(x(t)), \tag{2}$$

where $x(t) \in X \subset \mathbf{R}^n$. The function $f : X \longrightarrow \mathbf{R}^n$ is called a *vector field* on $\mathbf{R}^n$. We assume existence and uniqueness of solutions.[2]

Actually, the system of ODEs in (2) is called *autonomous* or *time invariant* because its vector field does not depend explicitly on time. If it did depend

[1] One can easily use difference equations instead. See [7].

[2] See [22] for conditions. A well-known *sufficient* condition is that f is *Lipschitz continuous*. That is, there exists $L > 0$ (called the *Lipschitz constant*) such that

$$\|f(x) - f(y)\| \leq L\|x - y\|, \qquad \text{for all } x, y \in X.$$

explicitly on time, it would be *nonautonomous* or *time varying*, which one might explicitly note using the following notation:

$$\dot{x}(t) = f(x(t), t). \tag{3}$$

An *ODE with inputs and outputs* [25,31] is given by

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t)), \\ y(t) &= h(x(t), u(t)), \end{aligned} \tag{4}$$

where $x(t) \in X \subset \mathbf{R}^n$, $u(t) \in U \subset \mathbf{R}^m$, $y \in Y \subset \mathbf{R}^p$, $f : \mathbf{R}^n \times \mathbf{R}^m \longrightarrow \mathbf{R}^n$, and $h : \mathbf{R}^n \times \mathbf{R}^m \longrightarrow \mathbf{R}^p$. The functions $u(\cdot)$ and $y(\cdot)$ are the *inputs* and *outputs*, respectively. Whenever inputs are present, as in (4), we say that $f(\cdot)$ is a *controlled vector field*.

Differential inclusions: A *differential inclusion* allows the derivative to belong to a set and is written as

$$\dot{x}(t) \in F(x(t)),$$

where $F(x(t))$ is a set of vectors in $\mathbf{R}^n$. It can be used to model nondeterminism, including that arising from controls or disturbances. For example, the controlled differential equation of (4) can be viewed as an inclusion by setting $F(x) = \cup_{u \in U} f(x, u)$.

Example 1 (Innacurate Clock). A clock that has a time-varying rate between 0.9 and 1.1 can be modeled by $\dot{x} \in [0.9, 1.1]$. Such an inclusion is called a *rectangular* inclusion. ◇

2.2 Adding discrete phenomena

We have seen that hybrid systems are those that involve continuous states and dynamics, as well as some *discrete phenomena* corresponding to discrete states and dynamics. As described above, our focus in this chapter is on the case where the continuous dynamics is given by a differential equation

$$\dot{x}(t) = \xi(t), \qquad t \geq 0. \tag{5}$$

Here, then, $x(t)$ is considered the *continuous component* of the hybrid state, taking values in some subset $\mathbf{R}^n$. The vector field $\xi(t)$ generally depends on $x(t)$ and the aforementioned discrete phenomena.

Hybrid *control* systems are those that involve continuous states, dynamics, and controls, as well as discrete phenomena corresponding to discrete states, dynamics, and controls. Here, $\xi(t)$ in (5) is a *controlled* vector field which generally depends on $x(t)$, the *continuous component* $u(t)$ of the control policy, and the aforementioned discrete phenomena.

In this section, we identify the discrete phenomena alluded to above that generally arise in hybrid systems. They are as follows:

- autonomous switching, where the vector field changes discontinuously;
- autonomous jumps, where the state changes discontinuously;
- controlled switching, where a control switches vector fields discontinuously;
- controlled jumps, where a control changes a state discontinuously.

Next, we examine each of these discrete phenomena in turn, giving examples.

Autonomous switching

Autonomous switching is the phenomenon where the vector field $\xi(\cdot)$ changes discontinuously when the continuous state $x(\cdot)$ hits certain "boundaries" [3, 28, 33, 37]. The simplest example of this is when it changes depending on a "clock" which may be modeled as a supplementary state variable [14]. Another example of autonomous switching is the hysteresis function from Section 1.

Example 2 (Furnace). Consider the problem of controlling a household furnace. The temperature dynamics may be quite complicated, depending on outside temperature, humidity, luminosity; insulation and layout; whether incandescent lights are on, doors are closed, vents are open, people are present; and many other factors. Thus, let's just say that when the furnace is On, the dynamics are given by $\dot{x}(t) = f_1(x(t))$, where $x(t)$ is the temperature at time t; likewise, when the furnace is Off, let's say that the dynamics are given by $\dot{x}(t) = f_0(x(t))$. The full system dynamics are that of a *switched system*:

$$\dot{x}(t) = f_{q(t)}(x(t)),$$

where $q(t) = 0$ or 1 depending on whether the furnace is Off or On, respectively. ◇

A particular class of hybrid systems of interest is *switched linear systems*:

$$\dot{x}(t) = A_{q(t)}x(t), \qquad q \in \{1, \ldots, N\},$$

where $x(t) \in \mathbf{R}^n$ and each $A_q \in \mathbf{R}^{n\times n}$. Such a system would be autonomous if the switchings were a function of the state $x(t)$. Sometimes the "switching rules" might interact with the constituent dynamics to produce unexpected results, as in the next example.

Example 3 (Unstable from Stable). Consider A_1 and A_2 where

$$A_1 = \begin{bmatrix} -0.1 & 1 \\ -10 & -0.1 \end{bmatrix}, \qquad A_2 = \begin{bmatrix} -0.1 & 10 \\ -1 & -0.1 \end{bmatrix}.$$

Then $\dot{x} = A_i x$, is globally exponentially stable for $i = 1, 2$. But the switched system using A_1 in the second and fourth quadrants and A_2 in the first and third quadrants is unstable. Fig. 7 plots ten seconds of trajectories for each of A_1, A_2 and the switched system starting from $(1, 0)$, $(0, 1)$, $(10^{-6}, 10^{-6})$, respectively. In each plot, motion is clockwise. In the first two, the range shown is $[-3, 3] \times [-3, 3]$; in the last, it is $[-2 \cdot 10^5, 8 \cdot 10^5] \times [-5 \cdot 10^4, 3.5 \cdot 10^5]$. ◇

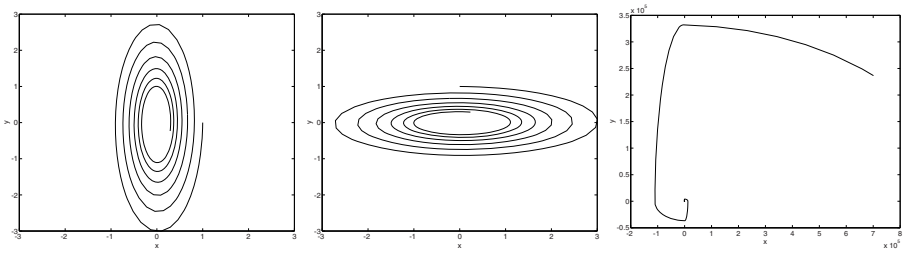

Fig. 7. Trajectories for (left) A_1x, (center) A_2x, and (right) a switched linear system

Autonomous jumps

An *autonomous jump* is the phenomenon where the continuous state $x(\cdot)$ jumps discontinuously on hitting prescribed regions of the state space [4,5]. We may also call these *autonomous impulses*. The simplest examples possessing this phenomenon are those involving collisions.

Example 4 (Bouncing Ball). Consider the case of the vertical motion of a ball of mass m under gravity with constant g. The dynamics are given by

$$\begin{aligned} \dot{x} &= v, \\ \dot{v} &= -mg. \end{aligned}$$

Further, upon hitting the ground (assuming downward velocity), we instantly set v to $-\rho v$, where $\rho \in [0, 1]$ is the coefficient of restitution. We can encode the jump in velocity as a rule by saying

If at time t, $x(t) = 0$ and $v(t) < 0$, then $v(t^+) = -\rho v(t)$.

In this case, $v(\cdot)$ is piecewise continuous (from the right), with discontinuities occurring when $x = 0$. This "rule" notation is quite general, but cumbersome. We have found it more desirable to use the following equational notation:

$$v^+(t) = -\rho v(t), \qquad (x(t), v(t)) \in \{(0, v) \mid v < 0\}$$

Here, we have used Sontag's evocative discrete-time transition notation [31] to denote the "successor" of $x(t)$. ◇

A general system subject to autonomous impulses may be written as

$$\begin{aligned} \dot{z}(t) &= f(z(t)), & z(t) &\notin A \\ z^+(t) &= G(z(t)), & z(t) &\in A. \end{aligned} \tag{6}$$

The interpretation of these equations is that the dynamics evolves according to the differential equation while z is in the complement of the *autonomous jump set* A, but that the state is immediately reset according to the map G upon z's hitting the set A.

Controlled switching

Controlled switching is the phenomenon where the vector field $\xi(\cdot)$ changes abruptly in response to a control command, usually with an associated cost. This can be interpreted as switching between different vector fields [38]. Controlled switching arises, for instance, when one is allowed to pick among a number of vector fields:

$$\dot{x} = f_q(x), \qquad q \in Q \simeq \{1, 2, \ldots, N\}.$$

Here, the q that is active at any given time is to be chosen by the controller. If one were to make the choice an explicit function of state, then the result would be a closed-loop system with autonomous switches.

Example 5 (Satellite Control). In satellite control, one encounters

$$\ddot{\theta} = \tau_{\text{eff}} v,$$

where θ, $\dot{\theta}$ are the angular position and velocity, respectively, and $v \in \{-1, 0, 1\}$, depending on whether the reaction jets are full reverse, off, or full on. ◇

Example 6 (Transmission). This example includes controlled switching and continuous controls. Consider a simplified model of a manual transmission, modified from one in [14]:

$$\begin{aligned} \dot{x}_1 &= x_2, \\ \dot{x}_2 &= [-a(x_2/v) + u]/(1 + v), \end{aligned}$$

where x_1 is the ground speed, x_2 is the engine RPM, $u \in [0, 1]$ is the throttle position, and $v \in \{1, 2, 3, 4\}$ is the gear shift position. The function a is positive for a positive argument. ◇

Controlled jumps

A *controlled jump* is the phenomenon where the continuous state $x(\cdot)$ changes discontinuously in response to a control command, usually with an associated cost [6]. We also call these jumps *controlled impulses*.

Example 7 (Inventory Management). In a simple inventory management model [6], there are a "discrete" set of restocking times $\theta_1 < \theta_2 < \cdots$ and associated order amounts $\alpha_1, \alpha_2, \ldots$. The equations governing the stock at any given moment are

$$\dot{x}(t) = -\mu(t) + \sum_i \delta(t - \theta_i)\alpha_i,$$

where μ represents degradation/utilization and δ is the Dirac delta. Note: If one makes the stocking times and amounts an explicit function of x (or t), then these controlled jumps become autonomous jumps. ◇

Example 8 (Planetary Flybys). Exploration spacecraft typically use close encounters with moons and planets to gain energy and change course. At the level of the entire solar system, these maneuvers are planned by considering the flight path to be a sequence of parabolic curves, with resets of heading and velocity occurring at the "point" of encounter. ◇

3 From Discrete to Hybrid

In this section, we review finite-state machines, or finite automata, as a base discrete model. We then successively add timing and continuous dynamics, arriving at models that approach more general hybrid systems.

3.1 Base discrete model: Automata

The base discrete model is usually a finite-state machine, finite transition system, or finite automaton, for which there is a very rich and beautiful theory [23]. These are actually discrete dynamical systems that process inputs. Therefore, the direct correspondent to the autonomous ODE of (2) would be an *inputless automaton*, which is a dynamical system with a discrete state space:

$$q_{k+1} = \nu(q_k),$$

where $q_k \in Q$, a finite set.

Example 9 (Finite Counter). A finite counter (modulo N) is an inputless automaton with $Q = \{0, 1, \ldots, N-1\}$ with $\nu(q) = q + 1 \mod N$. ◇

Preliminaries: Before defining automata with input, we begin with some standard material for discussing discrete inputs. A *symbol* is the abstract entity of automata theory. Examples are letters and digits. An *alphabet* is a finite set of symbols. For example, the English alphabet is $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$ and the binary alphabet is $B = \{\mathtt{0}, \mathtt{1}\}$. A *string* or a *word* (over alphabet I) is a finite sequence of juxtaposed symbols from I. Thus, `cat` and `jazz` and `zebra` are strings over A. So are `w` and `qqq`. The *empty string*, denoted ε, is the string consisting of zero symbols. The set of all strings over an alphabet I is denoted by I^*. The set of all binary strings is

$$B^* \equiv \{\varepsilon, \mathtt{0}, \mathtt{1}, \mathtt{00}, \mathtt{01}, \mathtt{10}, \mathtt{11}, \mathtt{000}, \mathtt{001}, \ldots\}.$$

A *language* (over alphabet I) is merely a set of strings over I. Thus, the words in a dictionary form a language over A, namely, the English language, $E \subset A^*$. Obviously, $\mathtt{cat} \in E$ and $\mathtt{qqq} \notin E$. The set of all binary strings of length two is a language over B, as is the set of all binary strings of even length. So is B^*. Yet another is the strings of odd parity (those having an odd number of ones):

$$B_{odd} \equiv \{\mathtt{1}, \mathtt{01}, \mathtt{10}, \mathtt{001}, \ldots\}.$$

The empty set is the *empty language*, namely the language consisting of no strings whatsoever. Note that it is different than the language $\{\varepsilon\}$.

DFA: We are now ready to define a *(deterministic) finite automaton* (*DFA*) which is a four-tuple $\mathcal{A} = (Q, I, \nu, q_0)$, where

- Q is a finite set of *states*,
- I is an alphabet, called the *input alphabet*,
- ν is the *transition function* mapping $Q \times I$ into Q, and
- $q_0 \in Q$ is the *initial state*.

The idea is that the machine above begins in state q_0 and then responds to or processes input strings over I. In one move, a DFA in state q receives/processes symbol $a \in I$ and enters state $\nu(q, a)$. On input word $w = a_1 a_2 \cdots a_n$, the DFA in state r_0 successively processes symbols and sequences through states $r_1, r_2, \ldots, r_n$, such that

$$r_{k+1} = \nu(r_k, a_{k+1}); \qquad k = 0, 1, 2, \ldots, n-1. \tag{7}$$

This sequence is called a *run* of the DFA over w.

Example 10 (Parity). Consider DFA $\mathcal{A}_{\text{par}} = (\{q_0, q_1\}, \{\mathtt{0}, \mathtt{1}\}, \nu, q_0)$, with

$$\nu(q_0, \mathtt{1}) = q_1; \qquad \nu(q_1, \mathtt{1}) = q_0; \qquad \nu(q_i, \mathtt{0}) = q_i,\ i \in \{0, 1\};$$

see Fig. 8(left). It keeps track of the parity of its input string by counting 1s, modulo 2. On input 111, it has run q_0, q_1, q_0, q_1; on ε, it has run q_0. ◇

NFA: The base definition of a DFA can be easily augmented in various ways. For example, a *nondeterministic finite automaton* (*NFA*) would allow a *set* of start states and a set-valued transition function mapping $Q \times I$ into 2^Q, so that at any stage the automaton may be in a set of states consistent with its transition function and the symbols seen so far. In one move, an NFA in state q receives/processes symbol a and nondeterministically enters any one of the states in $\nu(q, a)$. On input word $w = a_1 a_2 \cdots a_n$, an NFA in state r_0 nondeterministically sequences through states $r_1, r_2, \ldots, r_n$ such that

$$r_{k+1} \in \nu(r_k, a_{k+1}); \qquad k = 0, 1, 2, \ldots, n-1.$$

Again, such a sequence is a *run* of the NFA over w. In general, an NFA has many runs associated with each string; a DFA only has one.

Marked states: Actually, the traditional definitions of DFA and NFA add a set of marked or accepting or *final states*, F. In computation theory, one then defines the *language of A*, which is the set of strings *accepted* by the machine: the set of strings that have at least one run that ends in a state in F. For example, if $F = \{q_1\}$ in Example 10 above, 111 is accepted and ε is not. In fact, the language of $\mathcal{A}_{\text{par}}$ in this case is B_{odd}. If $F = \{q_0, q_1\}$, the accepted language would be B^*; if $F = \{q_1\}$, it would be $B^* - B_{\text{odd}}$; if $F = \emptyset$, it is $\emptyset$.

ω-Automata: Yet another way to augment the definition above is to allow machines that process infinite sequences of symbols. An *ω-string* (read "omega string") or *ω-word* over an alphabet I is an infinite-length sequence of symbols from I. For example, the following are ω-strings over B:

$$\mathtt{0}^\omega \equiv \mathtt{0000}\cdots \qquad \mathtt{1}^\omega \equiv \mathtt{1111}\cdots \qquad (\mathtt{01})^\omega \equiv \mathtt{0101}\cdots$$

Likewise, ω-languages are sets of ω-words. *ω-automata* act in exactly the same way as finite automata, and can be deterministic or not. Namely, on input ω-word $w = a_1 a_2 \cdots$, a DFA in state r_0 would successively process symbols and sequence through states $r_1, r_2, \ldots$, satisfying (7). Again, this sequence of states is a *run* over w. In Example 10, the run over 1^ω is $q_0, q_1, q_0, q_1, \ldots$.

More general automata: Finally, one can add discrete outputs and allow a countable (versus finite) number of states, resulting in a digital or symbolic or discrete automaton, with input and output. A *discrete automaton* is a machine (Q, I, ν, O, η) consisting of the state space, input alphabet, transition function, output alphabet, and output function, respectively. We assume that Q, I, and O are each countable. When these sets are finite, the result is a finite automaton with output. In any case, the functions involved are $\nu : Q \times I \longrightarrow Q$ and $\eta : Q \times I \longrightarrow O$. The *dynamics* of the automaton are specified by

$$\begin{aligned} q_{k+1} &= \nu(q_k, i_k), \\ o_k &= \eta(q_k, i_k). \end{aligned} \tag{8}$$

Such a model is easily seen to encompass finite automata and ω-automata, as well as other models from the literature (Mealy and Moore machines, push-down automata, Turing machines, Petri nets, etc.) [15, 23].

Example 11 (Mealy Machine). Consider the discrete automaton (derived from [23]; see Fig. 8) $\mathcal{A}_{\text{same}} = (\{q_\varepsilon, q_0, q_1\}, \{0, 1\}, \nu, \{\mathtt{n}, \mathtt{y}\}, \eta)$, with

$$\nu(q_\varepsilon, i) = q_i,\ \nu(q_i, 0) = q_0,\ \nu(q_i, 1) = q_1;\ \ \eta(q_\varepsilon, i) = \mathtt{n},\ \eta(q_i, i) = \mathtt{y},\ \eta(q_i, \bar{i}) = \mathtt{n};$$

each for $i \in \{0, 1\}$, with $\bar{0} = 1$ and $\bar{1} = 0$. If this machine starts in q_ε, it outputs `y` or `n` depending on whether the last two symbols seen are the same or not, respectively. Thus, on input `011`, it has run $q_\varepsilon, q_0, q_1, q_1$ and output `nny`. ◇

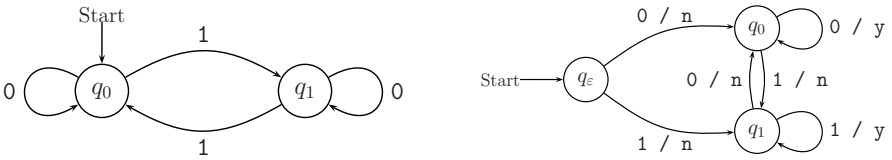

Fig. 8. (left) Finite automaton for parity, (right) example Mealy machine

3.2 Adding continuous phenomena

In moving toward hybrid systems, we may add a variety of more complex continuous phenomena to the finite and discrete automata above. The continuous phenomena we will add are as follows:

- *Global time*, where one adds a single global clock with unity rate.
- *Timed automata*, where one adds a set of such clocks, and the ability to reset them,
- *Skewed-clock automata*, where each clock has a different, uniform (in each location), rational rate.
- *Multi-rate automata*, where each clock variable can take on different, rational rates in each location.

Next, we examine each of these in turn. For ease of discussion, note that discrete states are also referred to as *modes*, *phases*, or *locations*.

Global time

Usually, digital automata are thought of as evolving in "abstract time," where only the ordering of symbols or "events" matters. See (8). We may add the notion of time by associating with the kth transition the time, t_k, at which it occurs [15]:

$$\begin{aligned} q(t_{k+1}) &= \nu(q(t_k), i(t_k)), \\ o(t_k) &= \eta(q(t_k), i(t_k)). \end{aligned}$$

Finally, this automaton may be thought of as operating in "continuous time" by the convention that the state, input, and output symbols are piecewise continuous functions. We may again use Sontag's transition notation [31] to denote this. The result is the following system of equations, where q, o are piecewise continuous in time:

$$\begin{aligned} q^+(t) &= \nu(q(t), i(t)), \\ o(t) &= \eta(q(t), i(t)). \end{aligned} \tag{9}$$

Here, the state $q(t)$ changes only when the input symbol $i(t)$ changes.

Timed automata

Timed automata more fully incorporate the notion of real time with automata [1]. Whereas finite automata process words, timed automata process *timed* words. A *timed word* (over input alphabet I) is a finite sequence of symbols and their respective times of occurrence: $(i_1, t_1), (i_2, t_2), \ldots, (i_N, t_N)$, where each $i_k \in I$, each $t_k \in \mathbf{R}_+$, and times *strictly increase*: $t_{k+1} > t_k$. A *timed ω-word* is an infinite sequence $(i_1, t_1), (i_2, t_2), \ldots$, with all as above plus the

constraint that time *progresses* without bound: for all $T \in \mathbf{R}_+$, there exists a k such that $t_k > T$. The latter condition is necessary to avoid *Zeno behavior*. For example, the sequence of times $1/2, 3/4, 7/8, 15/16, 31/32, \ldots$ is not a valid time sequence. A *timed language* is merely a set of timed words.

A *timed automaton* is the same as an automaton except that one adds a finite number of real-valued clocks, plus the ability to reset clocks and test constraints on clocks when traversing edges. A *clock constraint*, χ, may take one of the forms

$$(x \leq c), \qquad (c \leq x), \qquad \neg\chi_0, \qquad \chi_1 \wedge \chi_2,$$

where x is a clock variable, c is a rational constant, $\neg$ denotes logical negation, $\wedge$ denotes logical "and", and χ_i are themselves valid clock constraints. Note that these forms plus the rules of logic allow one to build up more complicated tests, including the following:

$$\begin{aligned}(x = c) &\Longleftarrow (x \leq c) \wedge (c \leq x),\\ (x < c) &\Longleftarrow (x \leq c) \wedge \neg(x = c),\\ \chi_1 \vee \chi_2 &\Longleftarrow \neg(\neg\chi_1 \wedge \neg\chi_2),\\ \mathsf{True} &\Longleftarrow (x \leq c) \vee (c \leq x).\end{aligned}$$

In drawings, the notation $?\chi$ is used to denote the fact that the edge *may* be taken only if the clock constraint χ is true; we have found it useful to let $!\chi$ mean that the edge *must* be taken when χ is true.

Example 12 (Bounded Response Time). Consider the timed automaton in Fig. 9(left), which is taken from [1]. If it starts in q_0, it will process strings of the form $(\mathtt{ad})^\omega$ such that the time between every `a` and its corresponding `d` is less than 2. It can be used to model the fact that every "arrival" needs to be serviced (needs to "depart") within two seconds. ◇

Example 13 (Switch with Delay). This example is modified from one describing a metal oxide semiconductor (MOS) transistor [26]. It can also be used to model relays, pneumatic valves, and other switches with delay. Specifically, consider a switch that can be On or Off, but that takes one second to activate. Let `U` and `D` denote the symbolic inputs corresponding to commanding the switch on or off, respectively. Then the switch can be modeled as a timed automaton as in Fig. 9(right). Note that issuing command `U` resets the clock to zero, and if a `D` command arrives within one time unit later, the switch goes back to off. ◇

The theory of timed automata is almost as rich and beautiful as the theory of finite automata. In fact, certain verification problems for timed automata can be translated directly into questions about (generally much larger) finite-state machines. See [1] for details.

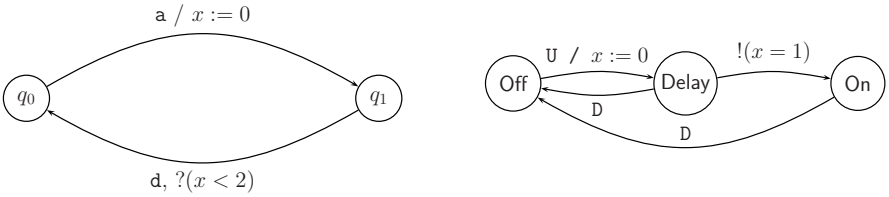

Fig. 9. Timed automata modeling (left) bounded response time, (right) a switch with delay

Skewed-clock automata

To summarize, a timed automaton has a finite set, C, of clocks, $x_i \in C$ such that $\dot{x}_i = 1$ for all clocks and all locations. In a *skewed-clock automaton*, $\dot{x}_i = k_i$ where each k_i is a rational number.

Remark 1. Skewed-clock automata are equivalent to timed automata. That is, any skewed-clock automaton can be converted into an equivalent timed automaton, and vice versa.

Proof. It is obvious that every timed automaton is a special case of a skewed-clock automaton, wherein each $k_i = 1$. For the converse, we have two cases to consider:

1. $k_i = 0$: In this case, $x_i(t)$ remains constant and any conditions involving it are uniformly true or false (and thus may be reduced or removed using the rules of logic).
2. $k_i \neq 0$: In this case, we note that $x_i(t) = x_i(0) + k_i t$, so that $x_i(t)/k_i = x_i(0)/k_i + t$. This means we can divide every constant to which x_i is compared by k_i, and then use the associated clock $\tilde{x}_i = x_i/k_i$, with $\dot{\tilde{x}}_i = 1$.

□

Multi-rate automata

A *multi-rate automaton* has $\dot{x}_i = k_{i,q}$ at location $q \in Q$, where each $k_{i,q}$ is a rational number; see Fig. 10.

Some notes are in order, both with respect to Fig. 10 and in general:

- Some variables might have the same rates in all states, e.g., w.
- Some variables might be *stopwatches* (derivative either 0 or 1), measuring a particular time. For example, x is a stopwatch measuring the elapsed time spent in the upper left state.
- Not all dynamics change at every transition. For example, y changes dynamics in transitioning along Edges 2 and 3, but not along Edge 1.
- Every skewed-clock automaton is a multi-rate automaton that simply has $k_{i,q} = k_i$ for all $q \in Q$.

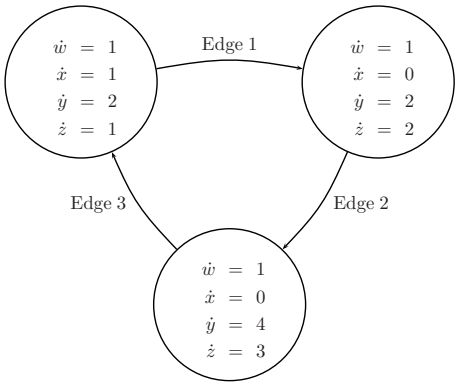

Fig. 10. Example multi-rate automaton (edge data has been suppressed)

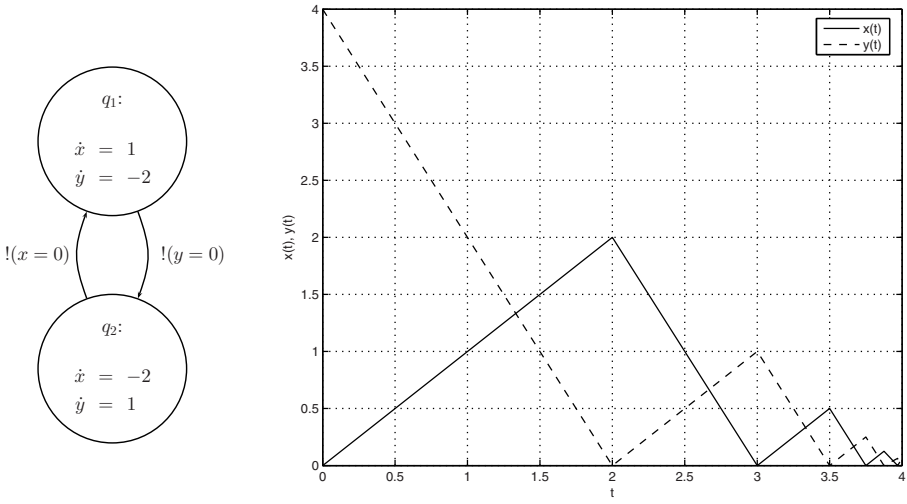

Fig. 11. (left) A multi-rate automaton with Zeno behavior. (right) Continuous-state dynamics over time, starting in q_1 at $(x, y) = (0, 4)$; events pile up at $t = 4$

Multi-rate automata can exhibit what is called *Zeno behavior* [39], where event times "pile up," not allowing time to progress; see Fig. 11. In general, the behavior of multi-rate automata is even more complicated, being computationally undecidable [21].

There is a simple constraint on multi-rate automata that yields a class that permits analysis. An *initialized* multi-rate automaton adds the constraint that a variable must be reset when traversing an edge if its dynamics change while crossing that edge. This was not the case for the Zeno system of Fig. 11. This would be the case for the automaton in Fig. 10 if x and z were initialized on Edge 1, y and z were initialized on Edge 2, and x, y, and z were initialized on Edge 3.

Remark 2. An initialized multi-rate automaton can be converted into a timed automaton.

Proof. The idea is to use the same trick as in Remark 1, as many times for each variable as it has different rates. Note that the fact that the automaton is "initialized" is crucial. See [21] for details. □

3.3 Other refinements

Rectangular automata: Going from rates to rectangular inclusions, one can define rectangular and multi-rectangular automata. Specifically, a *rectangular automaton* has each $\dot{x}_i \in [l_i, u_i]$, where l_i and u_i are rational constants, uniformly over all locations. Thus, they generalize skewed-clock automata. Indeed, letting $l_i = u_i = k_i$ recovers them. Analogously, multi-rate automata can be generalized to *multi-rectangular automata*. That is, each variable satisfies a (generally different) rectangular inclusion in each location. Usually, rectangular automata allow the setting of initial values and the resetting of variables to be performed within rectangles as well. *Initialized* multi-rectangular automata are constrained to perform a reset on a variable along edges that change the rectangular inclusion governing its dynamics.

Remark 3. An initialized multi-rectangular automaton can be converted to an initialized multi-rate automaton (and hence a timed automaton).

Proof. The idea is to replace each continuous variable, say x, with two variables, say x_l and x_u, that track lower and upper bounds on its value, respectively. See [21] for details. Then, invoke Remark 2. □

Adding control: A rich control theory for automata models can be built by allowing the disabling of certain *controllable* input symbols. See [15, 30].

4 Hybrid Dynamical Systems and Hybrid Automata

Briefly, a hybrid dynamical system is an indexed collection of ODEs along with a map for "jumping" among them (switching the dynamical system and/or resetting the state). This jumping occurs whenever the state satisfies certain conditions, given by its membership in a specified subset of the state space. Hence, the entire system can be thought of as a sequential patching together of ODEs with initial and final states, the jumps performing a reset to a (generally different) initial state of a (generally different) ODE whenever a final state is reached. See Fig. 12.

Formally, a *hybrid dynamical system* (*HDS*)[3] is a system $H = (Q, \mathbf{\Sigma}, \mathbf{A}, \mathbf{G})$, with constituent parts as follows:

[3] A more general notion, *GHDS* (*general HDS*), appears in [7]. It allows each ODE to be replaced by a general dynamical system; most notably, one could replace the ODEs with difference equations.

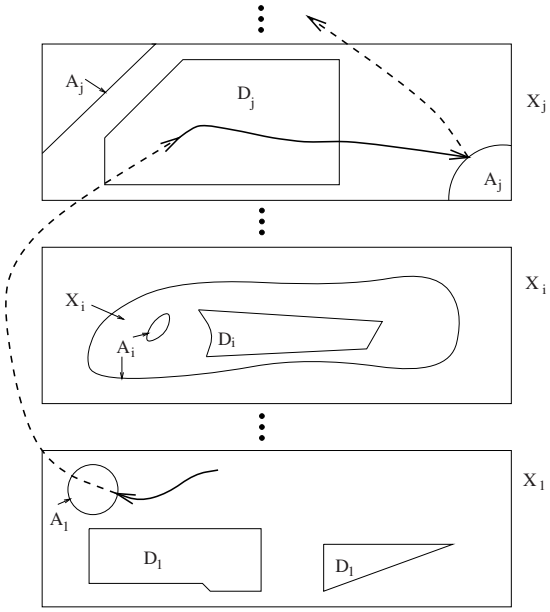

Fig. 12. Example dynamics of an HDS

- Q is the countable set of *index* or *discrete states*.
- $\mathbf{\Sigma} = \{\Sigma_q\}_{q\in Q}$ is the collection of *systems* of ODEs. Thus, each system Σ_q has vector fields $f_q : X_q \to \mathbf{R}^{d_q}$, representing the *continuous dynamics*, evolving on $X_q \subset \mathbf{R}^{d_q}$, $d_q \in \mathbf{Z}_+$, which are the *continuous state spaces*.
- $\mathbf{A} = \{A_q\}_{q\in Q}$, $A_q \subset X_q$ for each $q \in Q$, is the collection of *autonomous jump sets*.
- $\mathbf{G} = \{G_q\}_{q\in Q}$, where $G_q : A_q \to S$ are the *autonomous jump transition maps*, said to represent the *discrete dynamics*.

Here, $S = \bigcup_{q\in Q} X_q \times \{q\}$ is the *hybrid state space* of H. For convenience, we use the following shorthand. $S_q = X_q \times \{q\}$, and $A = \bigcup_{q\in Q} A_q \times \{q\}$ is *the* autonomous jump set. $G : A \longrightarrow S$ is *the* autonomous jump transition map, constructed componentwise in the obvious way. The *jump destination sets* $\mathbf{D} = \{D_q\}_{q\in Q}$ may be defined by $D_q = \pi_1[G(A) \cap S_q]$, where π_i is projection onto the ith coordinate. The *switching manifolds* or *transition manifolds*, $M_{q,p} \subset A_q$, are given by $M_{q,p} = G_q^{-1}(D_p, p)$, i.e., the set of states for which transitions from index q to index p can occur.

The dynamics of the HDS H are as follows.[4] The system is assumed to start in some hybrid state in $S\backslash A$, say $s_0 = (x_0, q_0)$. It evolves according to $\dot{x} = f_{q_0}(x)$ until the state enters—if ever—A_{q_0} at the point $s_1^- = (x_1^-, q_0)$. At

[4]For the solutions described to be well defined, it is sufficient that for all q, A_q is closed, $D_q \cap A_q = \emptyset$, and f_q is Lipschitz continuous. Note, however, that in general these solutions are not continuous in the initial conditions. See [7] for details.

this time it is instantly transferred according to transition map to $G_{q_0}(x_1^-) = (x_1, q_1) \equiv s_1$, from which the process continues.

We now collect some notes about HDS:

- *ODEs and automata.* First, note that in the case $|Q| = 1$ and $A = \emptyset$ we recover all differential equations. With $|Q| = N$ and each $f_q \equiv 0$, we recover finite automata.
- *Outputs.* It is straightforward to add continuous and discrete output maps for each constituent system to the above model.
- *Changing state space and overlaps.* The state space may change. This is useful in modeling component failures or changes in dynamical description based on autonomous—and later, controlled—events which change it. Examples include the collision of two inelastic particles or an aircraft mode transition that changes variables to be controlled [27]. We allow the X_q to overlap and the inclusion of multiple copies of the same space. This may be used, for example, to take into account overlapping local coordinate systems on a manifold [4]. It was also used in the hysteresis example.
- *Transition delays.* It is possible to model the fact that autonomous jumps may not be instantaneous by simply adding an *autonomous jump delay map*, $\Delta_a : A \times V \longrightarrow \mathbf{R}_+$. This map associates a (possibly zero) delay to each autonomous jump. Thus, a jump begins at some time, τ, and is completed at some later time $\Gamma \geq \tau$. This concept is useful as jumps may be introduced to represent an aggregate set of relatively fast, transitory dynamics. Also, some commands include a finite delay from issuance to completion. An example is closing a hydraulic valve.

It is easy to see that the case of HDS with $|Q|$ finite is a coupling of finite automata and differential equations. Indeed, an HDS can be pictured as a *hybrid automaton* as in Fig. 13. There, each node is a constituent ODE, with the index the name of the node. Each edge represents a possible transition between constituent systems, labeled by the appropriate condition for the transition's being "enabled" and the update of the continuous state (cf. [20]). The notation ![condition] denotes that the transition *must* be taken when enabled.

Example 14 (Bouncing Ball Revisited). We may draw a hybrid automaton associated with the bouncing ball of Example 4. The velocity resets are autonomous and must occur when the ball hits the ground. See Fig. 14. ◇

Example 15 (Furnace Revisited). Consider the furnace controller of Example 2, and a goal to keep the temperature around 23 degrees Celsius. To avoid inordinate switching around the setpoint (known as *chattering*), we implement the control with hysteresis as in Fig. 15(left). This controller will

1. Turn the furnace On when it is Off and the temperature falls below 21.
2. Turn the furnace Off when it is On and the temperature rises above 25.

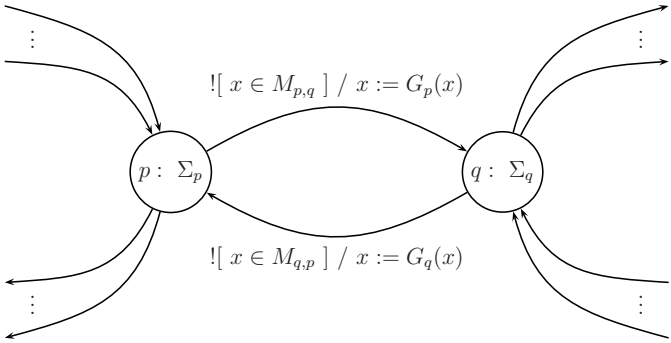

Fig. 13. Hybrid automaton representation of an HDS

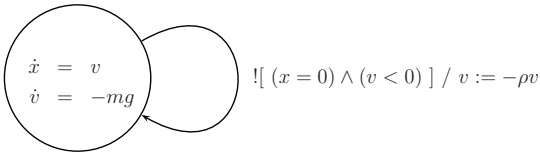

Fig. 14. Hybrid automaton for the bouncing ball of Example 4

Suppose, also, that the furnace should never be on more than 55 minutes straight, and that it must remain Off for at least 5 minutes. The latter can be accomplished by adding a timer variable that is reset on state transitions. The refined hybrid controller is pictured in Fig. 15(right). ◇

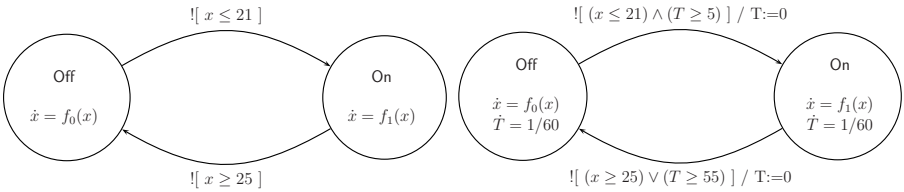

Fig. 15. Hybrid controllers for furnace: (left) simple hysteresis, (right) refinement

4.1 Adding control

We now add to the above the ability to make decisions, that is, to control an HDS by choosing among sets of possible actions at various times. Specifically, we allow (i) the possibility of continuous controls for each ODE, (ii) the ability to make decisions at the autonomous jump points, and (iii) the addition of controlled jumps, where one may decide to jump or not and have a choice of destination when the state satisfies certain constraints.

Formally, a *controlled hybrid dynamical system* (*CHDS*) is a system $H_c = (Q, \mathbf{\Sigma}, \mathbf{A}, \mathbf{G}, \mathbf{C}, \mathbf{F})$, with constituent parts as in an HDS, except as follows:

- $\mathbf{\Sigma} = \{\Sigma_q\}_{q\in Q}$ is now a collection of *controlled* ODEs, with controlled vector fields $f_q : X_q \times U_q \to \mathbf{R}^{d_q}$ representing the (controlled) *continuous dynamics*, where $U_q \subset \mathbf{R}^{m_q}$, $m_q \in \mathbf{Z}_+$, are the *continuous control spaces*.
- $\mathbf{G} = \{G_q\}_{q\in Q}$, where $G_q : A_q \times V_q \to S$ are the *autonomous jump transition maps*, now modulated by the *discrete decision sets* V_q.
- $\mathbf{C} = \{C_q\}_{q\in Q}$, $C_q \subset X_q$, is the collection of *controlled jump sets*.
- $\mathbf{F} = \{F_q\}_{q\in Q}$, where $F_q : C_q \to 2^S$, is the collection of set-valued *controlled jump destination maps*.

Again, $S = \bigcup_{q\in Q} X_q \times \{q\}$ is the hybrid state space of H_c. As before, we introduce some shorthand beyond that defined for the HDS above. We let $C = \bigcup_{q\in Q} C_q \times \{q\}$; we let $F : C \to 2^S$ denote *the* set-valued map composed in the obvious way from the set-valued maps F_q.

The dynamics of the CHDS H_c are as follows. The system is assumed to start in some hybrid state in $S\backslash A$, say $s_0 = (x_0, q_0)$. It evolves according to $\dot{x} = f_{q_0}(x, u)$ until the state enters—if ever—either A_{q_0} or C_{q_0} at the point $s_1^- = (x_1^-, q_0)$. If it enters A_{q_0}, then it *must* be transferred according to transition map $G_{q_0}(x_1^-, v)$ for some chosen $v \in V_{q_0}$. If it enters C_{q_0}, then we *may* choose to jump and, if so, we may choose the destination to be any point in $F_{q_0}(x_1^-)$. In either case, we arrive at a point $s_1 = (x_1, q_1)$ from which the process continues. See Fig. 16.

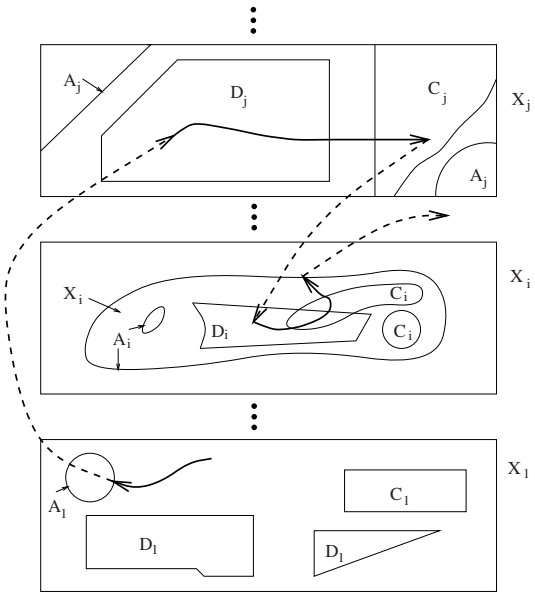

Fig. 16. Example dynamics of a CHDS

A CHDS can also be pictured as an automaton, as in Fig. 17. There, each node is a constituent controlled ODE, with the index the name of the node. The notation ?[condition] denotes an enabled transition that *may* be taken on command; ":∈" means reassignment to some value in the given set.

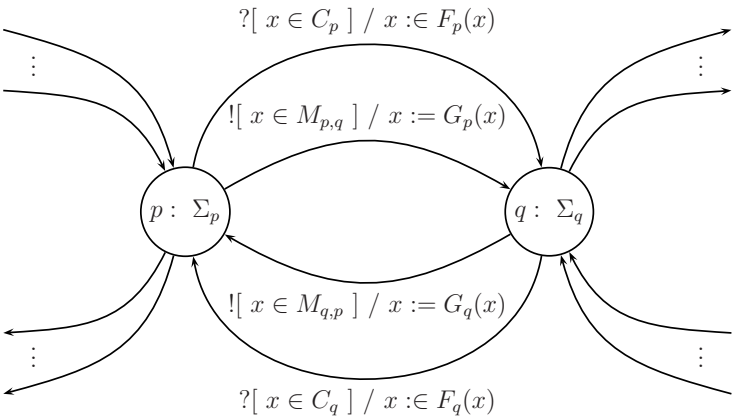

Fig. 17. Hybrid automaton representation of a CHDS

Example 16 (Transmission Revisited). Some modern performance sedans offer the driver the ability to shift gears electronically. However, there are often rules in place that prevent certain shifts, or automatically perform certain shifts, to maintain safe operation and reduce wear. These rules are often a function of the engine RPM (x_2 in Example 6). A portion of the hybrid controller incorporating such logic is shown in Fig. 18. The rules pictured only allow the driver to shift from Gear 1 to Gear 2 if the RPM exceeds 1800, but automatically makes this switch if the RPM exceeds 3500. Similar rules would exist for higher gears; more complicated rules might exist regarding Neutral and Reverse. ◇

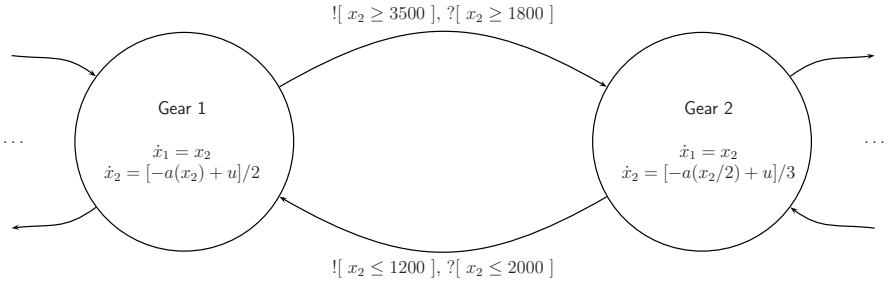

Fig. 18. Portion of hybrid transmission controller

5 Going Further

A survey of the hybrid systems literature is well beyond the scope of this chapter. For early surveys and more details on hybrid systems modeling, see [2,7,19]. For a recent monograph on switching systems, see [24]. Analysis and control techniques for hybrid systems have been developed. See [7] for details and [9] for a summary. For a more complete survey of stability tools, see [8,16]. The papers [10,11,13] developed an optimal control theory and algorithms for designing hybrid control systems. A game theoretic approach appears in [34]. For an introduction to hybrid systems simulation, see [12].

References

1. Alur, R., and Dill, D.L. (1994) A theory of timed automata. *Theoretical Computer Science*, 126:183–235.
2. Alur, R., Courcoubetis, C., Henzinger, T.A., and Ho, P.-H. (1993) Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R., Nerode, A., Ravn, A., and Rischel, H. (eds) (1993) *Hybrid Systems*, pp. 209–229. Springer, New York.
3. Antsaklis, P.J., Stiver, J.A., and Lemmon, M.D. (1993) Hybrid system modeling and autonomous control systems. In: Grossman, R., Nerode, A., Ravn, A., and Rischel, H. (eds) *Hybrid Systems*, pp. 366–392. Springer, New York.
4. Back, A., Guckenheimer, J., and Myers, M. (1993) A dynamical simulation facility for hybrid systems. In: Grossman, R., Nerode, A., Ravn, A., and Rischel, H. (eds) (1993) *Hybrid Systems*, pp. 255–267. Springer, New York.
5. Bainov, D.D., and Simeonov, P.S. (1989) *Systems with Impulse Effect.* Ellis Horwood, Chichester, England.
6. Bensoussan, A., and Lions, J.-L. (1984) *Impulse Control and Quasi-Variational Inequalities.* Gauthier-Villars, Paris.
7. Branicky, M.S. (1995) *Studies in Hybrid Systems: Modeling, Analysis, and Control.* ScD thesis, Massachusetts Institute of Technology, Cambridge, MA.
8. Branicky, M.S. (1997) Stability of hybrid systems: State of the art. In: *Proc. IEEE Conf. Decision and Control*, pp. 120–125, San Diego, CA.
9. Branicky, M.S. (1998) Analyzing and synthesizing hybrid control systems. In: Rozenberg, G., and Vaandrager, F. (eds) *Lectures on Embedded Systems*, pp. 74–113. Springer, Berlin.
10. Branicky, M.S., Borkar, V.S., and Mitter, S.K. (1998) A unified framework for hybrid control: Model and optimal control theory. *IEEE Trans. Automatic Control*, 43(1):31–45.
11. Branicky, M.S., Hebbar, R., and Zhang, G. (1999) A fast marching algorithm for hybrid systems. In: *Proc. IEEE Conf. Decision and Control*, pp. 4897–4902. Phoenix, AZ.
12. Branicky, M.S., and Mattsson, S.E. (1997) Simulation of hybrid systems. In: Antsaklis, P.J., Kohn, W., Nerode, A., and Sastry, S. (eds) *Hybrid Systems IV*, pp. 31–56. Springer, New York.
13. Branicky, M.S., and Mitter, S.K. (1995) Algorithms for optimal hybrid control. *Proc. IEEE Conf. Decision and Control*, pp. 2661–2666, New Orleans, LA.

14. Brockett, R.W. (1993) Hybrid models for motion control systems. In: Trentelman, H.L., and Willems, J.C. (eds) *Essays in Control*, pp. 29–53. Birkhäuser, Boston.
15. Cassandras, C.G., and Lafortune, S. (1999) *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Boston.
16. DeCarlo, R., Branicky, M.S., Pettersson, S., and Lennartson, B. (2000) Perspectives and results on the stability and stabilizability of hybrid systems. *Proc. IEEE*, 88(2):1069–1082.
17. Ghosh, R., and Tomlin, C. (2004) Symbolic reachable set computation of piecewise affine hybrid automata and its application to biological modelling: Delta-Notch protein signalling. *Systems Biology*, 1(1):170–183.
18. Gollu, A., and Varaiya, P.P. (1989) Hybrid dynamical systems. In: *Proc. IEEE Conf. Decision and Control*, pp. 2708–2712. Tampa, FL.
19. Grossman, R., Nerode, A., Ravn, A., and Rischel, H. (eds) (1993) *Hybrid Systems*. Springer, New York.
20. Harel, D. (1987) Statecharts: A visual formalism for complex systems. *Science Computer Programming*, 8:231–274.
21. Henzinger, T.A., Kopke, P.W., Puri, A., and Varaiya, P. (1998) What's decidable about hybrid automata? *J. Computer and System Sciences*, 57:94–124.
22. Hirsch, M.W., and Smale, S. (1974) *Differential Equations, Dynamical Systems, and Linear Algebra*. Academic Press, San Diego, CA.
23. Hopcroft, J.E., and Ullman, J.D. (1979) *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.
24. Liberzon, D. (2003) *Switching in Systems and Control*. Birkhauser, Boston.
25. Luenberger, D.G. (1979) *Introduction to Dynamic Systems*. Wiley, New York.
26. Maler, O., and Yovine, S. (1996) Hardware timing verification using KRONOS. In: *Proc. 7th Israeli Conf. Computer Systems and Software Eng.* Herzliya, Israel.
27. Meyer, G. (1994) Design of flight vehicle management systems. In: *IEEE Conf. Decision and Control*, Plenary Lecture. Lake Buena Vista, FL.
28. Nerode, A., and Kohn, W. (1993) Models for hybrid systems: Automata, topologies, controllability, observability. In: Grossman, R., Nerode, A., Ravn, A., and Rischel, H. (eds) *Hybrid Systems*, pp. 317–356. Springer, New York.
29. Raibert, M.H. (1986) *Legged Robots that Balance*. MIT Press, Cambridge, MA.
30. Ramadge, P.J.G., and Wonham, M.W. (1989) The control of discrete event systems. *Proc. IEEE*, 77(1):81–98.
31. Sontag, E.D. (1990) *Mathematical Control Theory*. Springer, New York.
32. Tabuada, P., Pappas, G.J., and Lima, P. (2001) Feasible formations of multi-agent systems. In: *Proc. Amer. Control Conf.*, Arlington, VA.
33. Tavernini, L. (1987) Differential automata and their discrete simulators. *Nonlinear Analysis, Theory, Methods, and Applications*, 11(6):665–683.
34. Tomlin, C., Lygeros, J., and Sastry, S. (2000) A game theoretic approach to controller design for hybrid systems. *Proc. IEEE*, 88(7):949–970.
35. Tomlin, C., Pappas, G.J., and Sastry, S. (1998) Conflict resolution for air traffic management: A study in multi-agent hybrid systems. *IEEE Trans. Automatic Control*, 43(4):509–521.
36. Varaiya, P.P. (1993) Smart cars on smart roads: Problems of control. *IEEE Trans. Automatic Control*, 38(2):195–207.
37. Witsenhausen, H.S. (1966) A class of hybrid-state continuous-time dynamic systems. *IEEE Trans. Automatic Control*, AC-11(2):161–167.

38. Zabczyk, J. (1973) Optimal control by means of switching. *Studia Mathematica*, 65:161–171.
39. Zhang, J., Johansson, K.H., Lygeros, J., and Sastry, S. (2000) Dynamical systems revisited: Hybrid systems with Zeno executions. In: Lynch, N., and Krogh, B. (eds) *Hybrid Systems: Computation and Control*, pp. 451–464. Springer, Berlin.
40. Zhang, W., Branicky, M.S., and Phillips, S.M. (2001) Stability of networked control systems. *IEEE Control Systems Magazine*, 21(1):84–99.