Structure and Interpretation of Signals and Systems
Edward Lee and Pravin Varaiya
ISBN: 0-201-74551-8

Chapter 6

Hybrid Systems

This text models signals and systems as functions. To develop understanding, we study the structure of the domain and range of these functions, as well as the structure of the mapping from the domain to the range. Despite the uniformity of this approach, we have begun to evolve two distinct families of models. Chapters 3 and 4 structure this mapping using state machines. Chapter 5 generalizes these state machines so that the number of possible states is infinite, and specializes them so that the systems are linear and time-invariant (LTI). LTI systems prove to yield to powerful analytical techniques, which are only hinted at in chapter 5. Chapters 7 through 11 will further develop these analytical techniques by structuring the system mapping using frequency-domain concepts.

The analytical methods available for LTI systems prove so compelling that we wish to apply them even to systems that are not LTI. In fact, no real-world system is truly LTI. At a minimum, its properties were certainly different during the initial stages of the big bang, so it cannot be time invariant. More practically, systems change over time; they are turned on and off, they deteriorate, etc. Moreover, systems that behave as linear systems typically do so only over some regime of operation. For example, if the magnitude the inputs exceed some threshold, a real-world system will overload, and will no longer behave linearly. A similar effect might result when the state wanders beyond some modest range. This chapter shows how models that are only applicable some of the time can be used effectively.

In chapters 3 and 4, signals are sequences of events. Their domain is (typically) $\mathit{Naturals}_0$, and their range is (typically) a finite and arbitrary set of symbols. The domain is not interpreted as time, but rather as indexes of a sequence. In chapters 5 and 7 through 11, the domain of signals is interpreted as time. For continuous-time signals, the domain is either *Reals* or Reals_+, whereas for discrete-time signals it is either *Integers* or $\mathit{Naturals}_0$, but in either case, the domain is interpreted as representing as advancing time. This interpretation of the domain as time is essential to the notion of frequency that is used throughout the forthcoming chapters.

Chapter 5 and this one provide a bridge between **state-machine models** and such **time-based models** by developing state machine models for time-based systems. In this chapter, we build another bridge between these two families of models by showing that they can often be usefully combined and used *simultaneously* in the *same* model, rather than as alternative views of a system. The resulting models are called **hybrid systems**. They are a powerful tool for understanding real-world

systems.

To understand the value of hybrid systems, it is useful to reflect on the relative strengths and weaknesses of time-based models and state-machine models. Chapter 5 demonstrates that state-machine models are more general by showing how they can be used to describe time-based models. Since they are more general, why not just always use state-machine models? The methods of state-machine models, such as composition by forming a product of the state spaces, simulation, and bisimulation, do not yield the depth of understanding that we will get in the subsequent chapters from looking at frequency response. Why not always use frequency response? Frequency response is a rather specialized analytical tool. It applies only to LTI systems. Most real-world systems are not LTI, so such powerful analytical tools must be applied with careful caveats about the regime of operation over which they do apply.

Consider for example a home audio system. It takes data from a compact disc and converts it into auditory stimulus. Is it LTI? Well, obviously not, since its system function changes rather drastically when you turn it on and off. The acts of turning it on and off, however, seem to match well the state transitions of a state machine. Can we come up with a model where there is a state machine with two states, "on" and "off," and associated with each state there is an LTI system that describes the behavior of the system in the corresponding mode of operation? Indeed we can. Such a model is called a hybrid system.

In order to get state machine models to coexist with time-based models, we need to interpret state transitions on the time line used for the time-based portion of the system, be it continuous time or discrete time. In the audio system, for example, we need to associate a time with the acts of turning it on or off. The models used in chapters 3 and 4 do not naturally do this, since the signals there are sequences of events. That is, they are functions whose domain is $\mathit{Naturals}_0$, where there is no temporal association with an $n \in \mathit{Naturals}_0$.

Recall from chapter 3 that the input and output alphabets of a state machine are required to include a stuttering element, typically denoted *absent*. Whenever the state machine reacts, if its input is the stuttering element, then it does not change state and its output will be the stuttering element. This is key to hybrid system models because it allows us to embed the state machine into a time-based model. At any time where there is no interesting input event, the machine stutters.

A hybrid system combines time-based signals with sequences of events. The time-based signals are of the form $x: T \to R$, where R is some range (such as *Reals* or *Complex*), and T is either *Reals*, Reals_+, *Integers*, or $\mathit{Naturals}_0$, depending on whether the time domain is discrete or continuous and whether the model includes a time origin. In chapters 3 and 4, the event signals had the form $u: \mathit{Naturals}_0 \to \mathit{Symbols}$, where the set *Symbols* has a stuttering element. For a hybrid system, however, these have to share a common time base with the time-based signals, so they have the form $u: T \to \mathit{Symbols}$. Thus, events occur in time. Typically, for most $t \in T$, $u(t) = \mathit{absent}$, the stuttering element. The non-stuttering element is used only at those discrete values of time where an event occurs.

6.1 Mixed models

A state machine model becomes a time-based model if it reacts at all times in the time base T. This means that state machines and time-based models can interact as peers, sending time-based signals to one another.

Example 6.1: Moving averages are popular on Wall Street for detecting trends in stock prices. But in using them, a key question arises: how long should the moving average be? A short-term moving average might detect short-term trends, while a long-term moving average might detect long term trends. A classical method combines the two and compares them to generate buy and sell signals. If the short-term trend is more sharply upward than the long term trend, a buy signal is generated. If the short term trend is more sharply downward than the long term trend, a sell signal is generated.

A system implementing this **moving average cross-over method** is shown in figure 6.1. The input is the discrete-time signal *price*:*Integers* $\rightarrow$ *Reals* representing the closing price of a stock each day. The LTI systems *shortTerm* and *longTerm* are both moving average systems, but *shortTerm* averages fewer successive inputs than *longTerm*. The outputs of these systems are the discrete-time signals x and y. The finite state machine reacts on each sample from these signals. It begins in the state *short over long*. The transition out of this state has the guard

$$\{(x(n), y(n)) \mid x(n) > y(n)\}.$$

When this transition is taken, a *buy* signal is generated. The sell signal is generated similarly. The plots below show the buy and sell signals generated by a (synthetic) sequence of stock prices.

This example illustrates a simple form of technical stock trading. In this extreme form, it has the controversial feature that it ignores the fundamentals of the company whose stock is being traded. It is using the stock price alone as the indicator of worth. In fact, much more sophisticated signal processing methods are used by technical stock traders, and they often do take as inputs other quantifiers of company worth, such as reported revenues and profits.

6.2 Modal models

In the previous section, time-based systems are combined with state machines as peers. A richer interaction is possible with a hierarchical combination. The general structure of a hierarchical hybrid system model is shown in figure 6.2. In that figure, there is a two-state finite state machine. There are some changes to the notation, however, from what was used in chapters 3 and 4.

First, notice that the inputs and outputs include both event signals and time-based signals. Second, notice that each state of the state machine is associated with a time-based system, called the **refinement** of the state. The refinement of a state gives the time-based behavior of *HybridSystem*

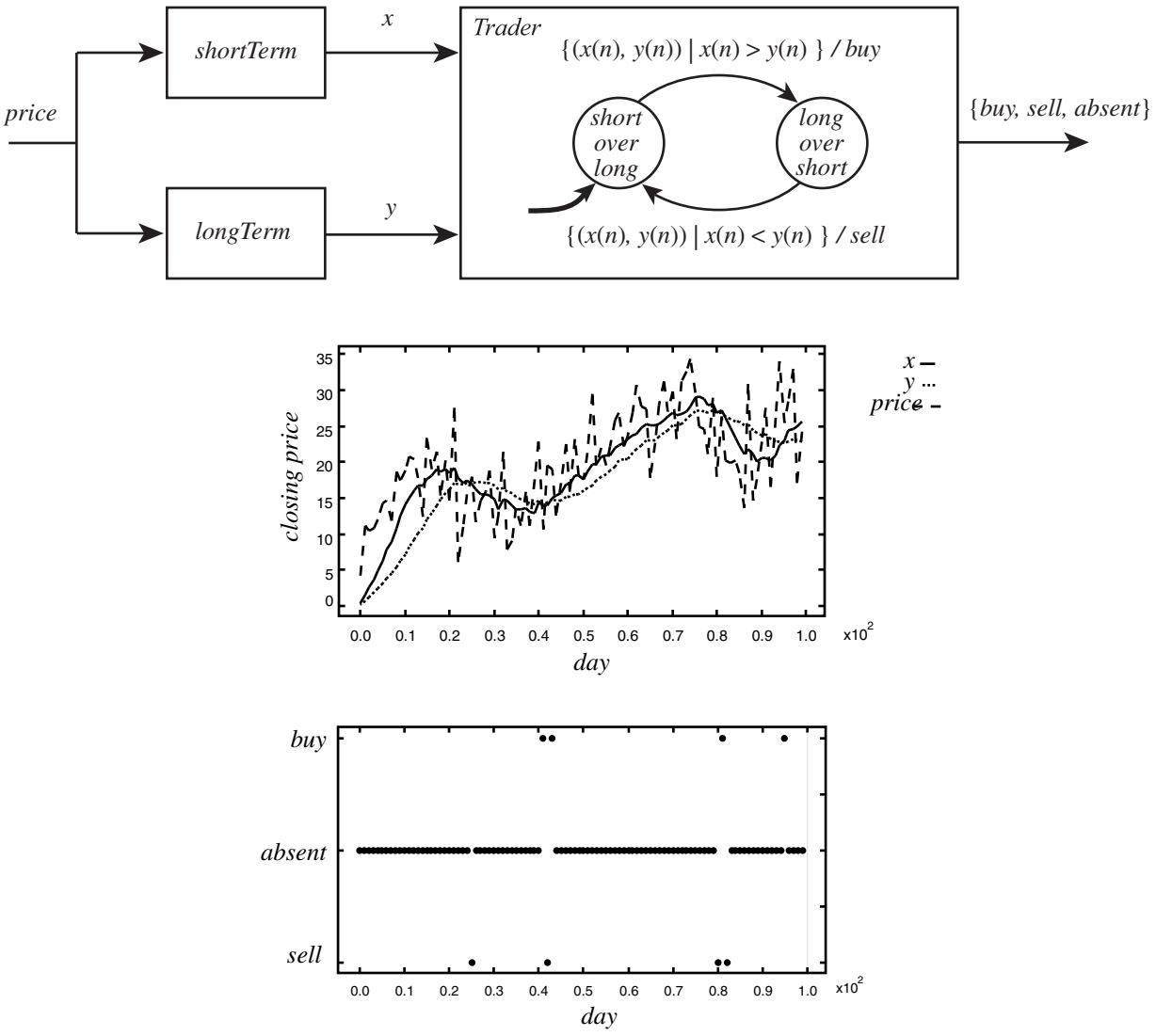


Figure 6.1: An implementation of the classical moving average cross-over method for trading stocks.

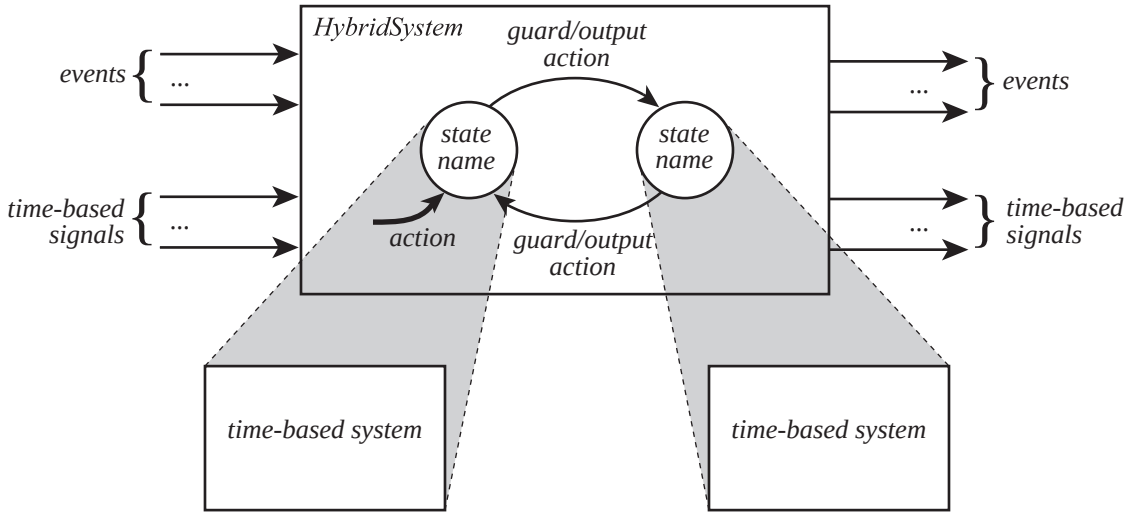


Figure 6.2: Notation for hybrid systems.

while the machine is in that state. Thus, the states of the state machine define **modes** of operation of the system, where the behavior in a given mode is given by the refinement. A hybrid system is sometimes called a **modal model** for this reason. The refinement has access to all the inputs of *HybridSystem*, and produces the time-based output signals of *HybridSystem* while the machine is in its mode.

Note that the term "state" for such a hybrid system can become confusing. The state machine has states, but so do the refinement systems (unless they are memoryless). When there is any possibility of confusion we explicitly refer to the states of the machine as **modes**, and we refer to the states of the refinement as **refinement states**. The (complete) state of the hybrid system is a pair (m, s) where m is the mode and s is the state of the time-based refinement system associated with mode m.

Another difference from the notation used in chapters 3 and 4 is that state transitions in the machine have, in addition to the usual guard and output notations, an **action**. The action will typically set the initial refinement state of the time-based system in the destination mode.

The guards are, as usual, sets. However, we need for the guards to be rich enough that a transition can be triggered by a particular value of a refinement state or by a value of a time-based input. Thus, the elements of the guards are tuples containing values of input events, time-based signals, and the refinement states. In the state machines in chapters 3 and 4, the elements of the guards only contained values of input events. For hybrid systems, we add time-based signals and refinement states.

> **Example 6.2:** Overload of an electronic system might be modeled by a state transition that is triggered by the magnitude of the current refinement state exceeding some threshold.

On the other hand, when the system is in some mode, the refinement state is only affected by the

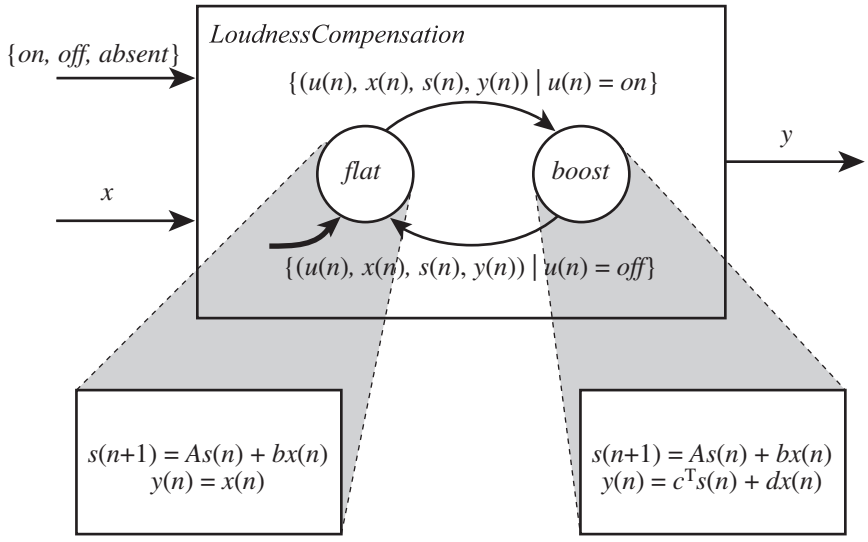


Figure 6.3: This system implements loudness compensation, described in example 6.3

time-based inputs. It is not affected by the event inputs. This keeps the time-based models simple, so that they don't have to deal with stuttering inputs.

Correspondingly, the time-based outputs are generated by the refinement, and hence need not be mentioned after the slash on the transitions.

The state machine may react at any time in the time base T. The mode in which it is before this reaction is called the **current mode**. It will take a discrete state transition and switch to the **destination mode** if the input values and the refinement state at that time match a guard. If it does not take a discrete state transition, then the state machine stutters. In either case, the refinement of the current mode also reacts to the time-based inputs, changes its state and produces outputs.

Example 6.3: Many high-end audio systems offer "digital signal processing." Such a system typically has an embedded computer (a digital signal processor or DSP, see box on page 318). This computer is used to process the audio signal in various ways, for example to add reverberation or to perform frequency selective filtering. A particularly simple function that might be performed is **loudness** compensation, something offered by all but the cheapest audio systems.

At low volumes, the human ear is less sensitive to low frequencies (base notes) than to high frequencies. Loudness compensation boosts the low frequencies. This is done simply by implementing a filter, which is a linear time-invariant system that can be described by a state-space model, as in the previous chapter. Thus, there are two modes, one where the low frequencies are boosted (using the filter), and one where they are not.

A simple realization of loudness compensation offers a switch on a control panel to turn on and off the compensation. Figure 6.3 shows a hybrid system that reacts to input events from this switch to select from among two modes. The upper input is simply an event indicating the position of the control switch when it is thrown. The lower input x is a discrete-time signal, probably sampled at 44,100 samples/second, the CD rate. The *LoudnessCompensation* hybrid system has two modes. In the *flat* mode, the output y is simply set equal to the input x. That is, if $T_{flat} \subset Integers$ is the time indexes during which the machine is in the *flat* mode, then

$$\forall\, n \in T_{flat}, \quad y(n) = x(n).$$

This (obviously) does not boost low frequencies, since the output is equal to the input.

When the *on* event occurs, the machine transitions to the *boost* mode, where the filter is applied to the input x. This is done using the state update and output equations

$$\begin{aligned} \forall\, n \in T_{boost}, \quad s(n+1) &= As(n) + bx(n) \\ y(n) &= c^T s(n) + dx(n), \end{aligned}$$

where A, b, c, d are chosen to boost the low frequencies (how to do that is explained in chapter 9).

Note that in the *flat* mode, even though the output equation does not depend on the state, the state update equation is still applied. This ensures that when switching between states, no glitches are heard in the audio signal. The state of the *boost* refinement is maintained even when the mode is *flat*.

This loudness compensator is not very sophisticated. A more sophisticated version would have a set of compensation filters and would select among them depending on the volume level. This is explored in exercise 1.

We consider a sequence of special cases of hybrid systems. Although the next few examples are all continuous-time models, it is easy to construct similar discrete-time models.

6.3 Timed automata

Timed automata are the simplest continuous-time hybrid systems. They are modal models where the time-based refinements have very simple dynamics; all they do is measure the passage of time. Such refinements are called **clocks**. The resulting models are finite state machines (automata) with time. Note that although all the examples in this section use continuous time, discrete-time versions are very similar.

A clock is modeled by a first-order differential equation,

$$\forall\, t \in T_m, \quad \dot{s}(t) = a,$$

where s: *Reals* $\rightarrow$ *Reals* is a function, $s(t)$ is the value of the clock at time t, and $T_m \subset T$ is the subset of time during which the hybrid system is in mode m. The rate of the clock, a, is a constant while the system is in this mode.

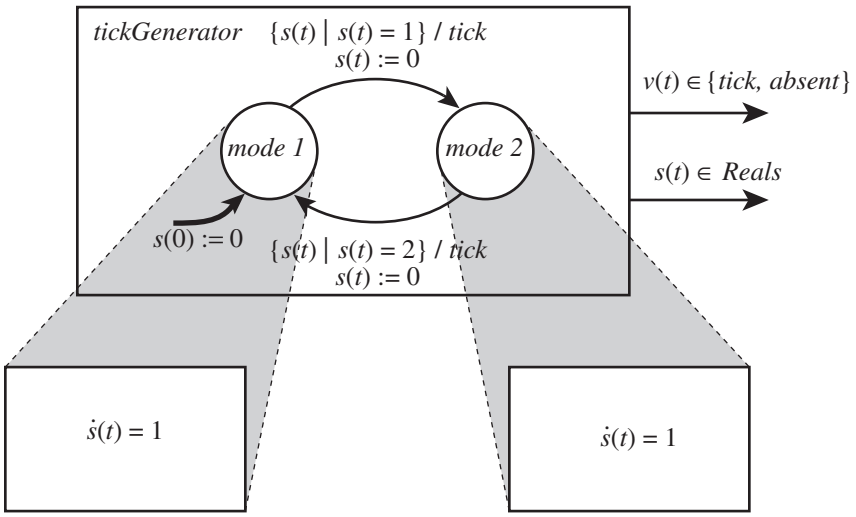


Figure 6.4: This hybrid system generates *tick* at time intervals alternating between 1 and 2 seconds. It is a timed automaton.

Example 6.4: Suppose we want to produce a sequence of output events called *tick* with the time between two consecutive *tick*s alternating between 1 and 2 seconds. That is, we want to produce a *tick* at times $1, 3, 4, 6, 7, 9, \cdots$.

A hybrid system *tickGenerator* that does this is illustrated in figure 6.4. There are two modes labeled *mode 1* and *mode 2*. The refinement state in each mode is the value of a clock at time t, denoted by $s \in \mathit{Reals}$. So at any time t the state of *tickGenerator* is the pair $(\mathit{mode}(t), s(t))$. The output is the event signal v and the time-based signal s. There is no input.

In both modes, s evolves according to the differential equation $\dot{s}(t) = 1$, where $\dot{s}(t)$ is the derivative of s with respect to time evaluated at some time t. Thus, s simply measures the passage of time, with its value rising 1 second for every second of elapsed time.

The behavior of the system is shown in figure 6.5. At time 0, as indicated by the bold arrow in figure 6.4, the system initially enters *mode 1*. The bold arrow has an action, "$s(0) := 0$," which sets $s(0)$ to 0. The notation ":=" is used instead of "=" to emphasize that this is an assignment, not an assertion (see section A.1.1).

In this example, there is no input, so a guard is a subset of the possible values (*Reals*) of the refinement states. The guard on the transition from *mode 1* to *mode 2* is

$$\{s(t) \mid s(t) = 1\},$$

which is satisfied one time unit after beginning. For all $t \in [0, 1]$, $s(t) = t$. At time $t = 1$, this guard is satisfied, the transition is taken, and the output event $v(1) = \mathit{tick}$ is produced. For all $t \in [0, 1)$, $v(t)$ has value *absent*.

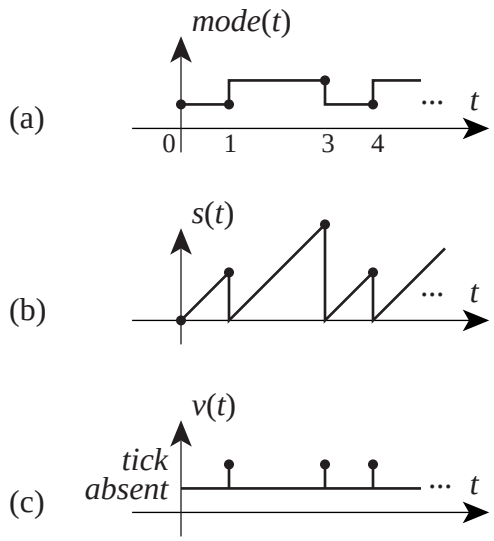


Figure 6.5: (a) The modes of the hybrid system of figure 6.4, (b) the refinement state s, and (c) the discrete event output v.

This transition also has an action, "$s(t) := 0$," which resets s to zero. This gives the initial condition for the refinement system of the destination mode. In our definition, at time $t = 1$, $s(t) = 1$, even though the action seems to contradict this. This is emphasized in figure 6.5 by showing with a bold dot the value of s at each discontinuity. The action $s(t) := 0$ is merely providing the initial conditions for the refinement of the destination mode. But the destination mode is not active until $t > 1$, so the action is setting $s(1+)$ to 0, where $1+$ denotes a time infinitesimally larger than 1.

For $t \in (1,3]$, the system remains in *mode 2*, evolving according to the differential equation

$$\begin{aligned} \dot{s}(t) &= 1, \\ s(1) &= 0. \end{aligned}$$

So for $1 < t \leq 3$,

$$s(t) = s(1) + \int_1^t 1 dt = t - 1.$$

At time $t = 3$, the guard on the arc from *mode 2* to *mode 1* is satisfied, so the transition is taken. The output event *tick* is again produced, and s is reset to 0 again.

Notice in figure 6.5 that the output v is *absent* for all but a few discrete values of $t \in Reals$. This signal is called a **discrete event** signal for this reason. Of course, this signal can also be reinterpreted as a sequence of *tick* events with an arbitrary number of stuttering events in between. That signal could therefore be supplied as input to an ordinary state machine, enabling compositions of ordinary state machines with hybrid systems.

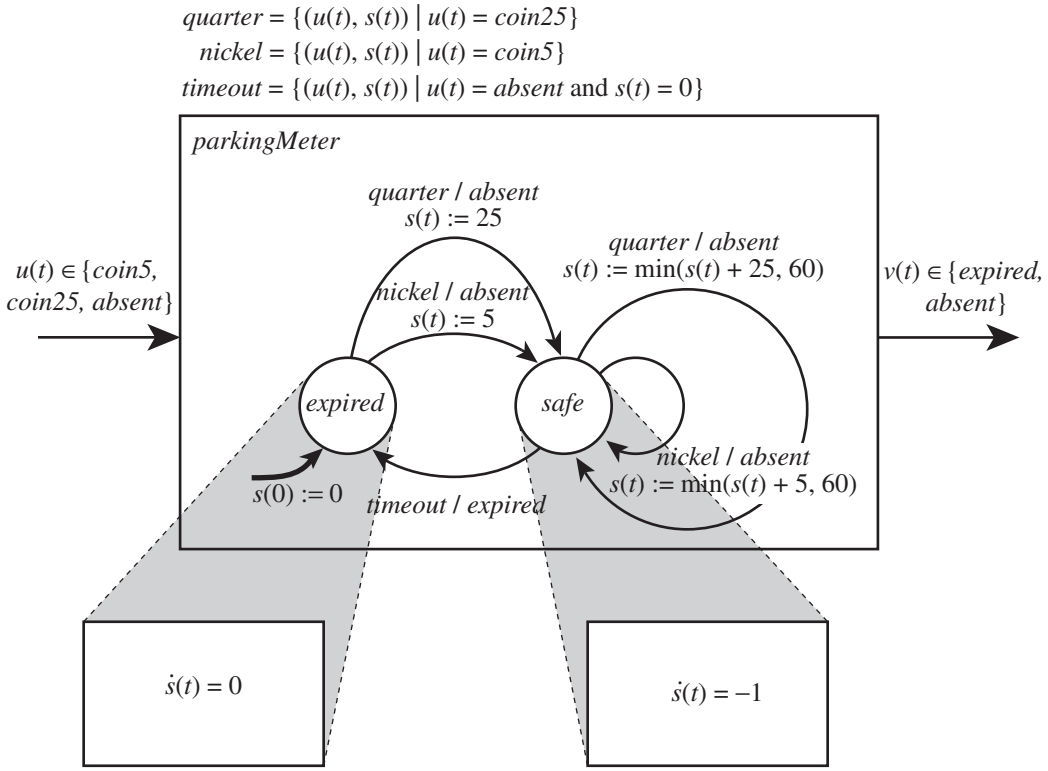


Figure 6.6: A hybrid system representation of a 60-minute parking meter.

Also notice in figure 6.5 that the hybrid system evolves in alternating phases: there is a **time-passage phase** in which the system stays in the same mode and its refinement state changes with the passage of time; this is followed by an instantaneous **discrete-event phase** in which a mode transition occurs, an output event is produced, and the refinement state in the destination mode is initialized. In the figure, the time-passage phases are $(0,1],(1,3],(3,4],\cdots$ and the discrete-event phases occur at $1,3,4,\cdots$.

Transitions between modes have actions associated with them. Sometimes, it is useful to have transitions from one mode back to itself, just so that the action can be realized. This is illustrated in the next example.

Example 6.5: Figure 6.6 shows a hybrid system representation of the 60-minute parking meter considered in chapter 3. In the version in figure 3.6, the states of a state machine are used to measure the passage of time by counting ticks provided by the environment. In the hybrid version of figure 6.6, the passage of time is explicitly modeled by first-order differential equations.

There are two modes, *expired* and *safe*, and the refinement state at time t is $s(t) \in Reals$. At time $t = 0$, the initial mode is *expired*, and $s(0) = 0$. In the *expired* mode, s remains at 0. The input events *coin5* and *coin25* cause one of two transitions from *expired* to

safe to be taken. These transitions have guards that are named *nickel* and *quarter* and are defined by

$$nickel = \{(u(t), s(t)) \mid u(t) = coin5\}$$

$$quarter = \{(u(t), s(t)) \mid u(t) = coin25\}.$$

Using names for these guards in the figure makes it more readable. It would be cluttered if the guards were directly noted on the transitions.

The transitions from *expired* to *safe* produce *absent*. The actions on the transitions change the value of s to 5 and 25, depending on whether *coin5* or *coin25* is received.

In the *safe* mode, the refinement state decreases according to the differential equation of the clock,

$$\forall\, t \in T_{safe}, \quad \dot{s}(t) = -1.$$

There are three possible outgoing transitions from this mode. If the input event *coin5* or *coin25* occurs, then one of two self-loop transitions is taken, no output is produced, and the associated action increments s by setting as $s(t) := \min(s(t) + 5, 60)$ or $s(t) := \min(s(t) + 25, 60)$. But if the guard *timeout* is satisfied, where

$$timeout = \{(u(t), s(t)) \mid u(t) = absent \text{ and } s(t) = 0\}$$

then there is a transition to *expired* and the output event *expired* is produced. Note that this guard requires that $u(t) = absent$, so that if the parking meter expires at the very moment that a coin arrives, then the coin is properly registered.

In this system, the refinement state evolves differently in the two modes; in *expired*, s remains at 0 (since $\dot{s}(t) = 0$), but in *safe*, s obeys the differential equation $\dot{s}(t) = -1$.

In the previous example, the transitions from *safe* back to *safe* were used for their actions, which react to input events by setting the values of refinement states. This gives a clean way to model discontinuities in continuous-time signals, because the state trajectory is a continuous-time signal. A more extreme example is given next, where there is only one mode.

Example 6.6: We could also implement the parking meter as a cascade composition using a timed automaton, *TickGenerator*, with only one mode, *timer*, and which produces a *tick* event every minute. This event serves as an input to the parking meter finite state machine of figure 3.6. The cascade composition is shown in figure 6.7. The parking meter machine also accepts an additional (product-form) input event from {*coin5*, *coin25*}, and produces the output event *safe* or *expired*. The difference between figure 3.6 and 6.7 is that in the former *tick* was an input event from the environment, whereas in the latter we explicitly construct a component, namely *TickGenerator*, which produces a *tick* every minute.

Timed automata are commonly used in modeling **communication protocols**, the logic used to achieve communication over a network. The following example models the transport layer of a sender of data on the internet.

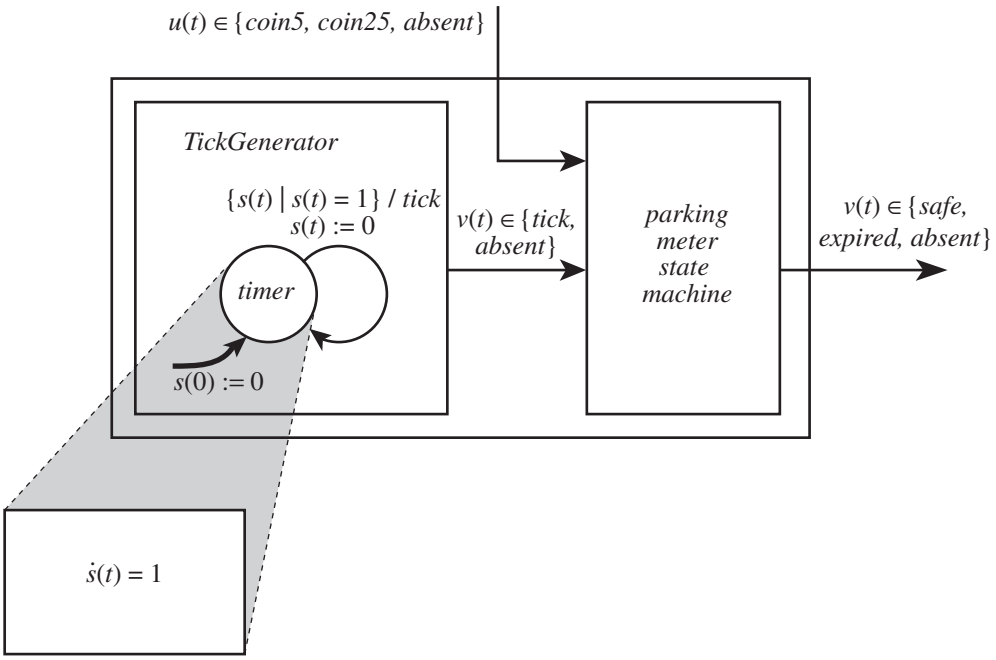


Figure 6.7: The 60-minute parking meter as a cascade composition of *tickGenerator* and an ordinary finite state machine.

Example 6.7: Consider how an application such as an e-mail program sends a file over a communication network like the internet. There are two host computers called the *Sender* and *Receiver*. The file that *Sender* wants to send to *Receiver* is first divided into a sequence of finite bit strings called **packets**. For the purposes of this example, we do not care what is contained by the packets, so we consider *packet* to be an event. We are interested in the fact that it needs to be transmitted, not in its contents.

The problem we address in this example is that the network is unreliable. Packets that are launched into it may never emerge. If the network is congested, packets get dropped. We will design a protocol whereby the sender of a packet waits a certain amount of time for an acknowledgement. If it does not receive the acknowledgement in that time, then it retransmits the packet. This is an ideal application for timed automata.

The upper diagram in figure 6.8 shows the structure of the communication system. Everything begins when the sender produces a *packet* event. The *SenderProtocol* system reacts by producing a *transmit* event, which instructs its **network interface card** or **NIC** to launch the packet into the internet. The NIC is the physical device (such as the ethernet card in your desktop computer) that converts the packet into the appropriate electrical signal that is transmitted through the network. The internet transfers this signal to the NIC of the receiver. That NIC converts the signal back into the packet and forwards it to the *ReceiverProtocol* component. The *ReceiverProtocol* in turn forwards the packet to the e-mail application in the *Receiver* and simultaneously sends an acknowledgement packet, called *ack*, to its NIC.

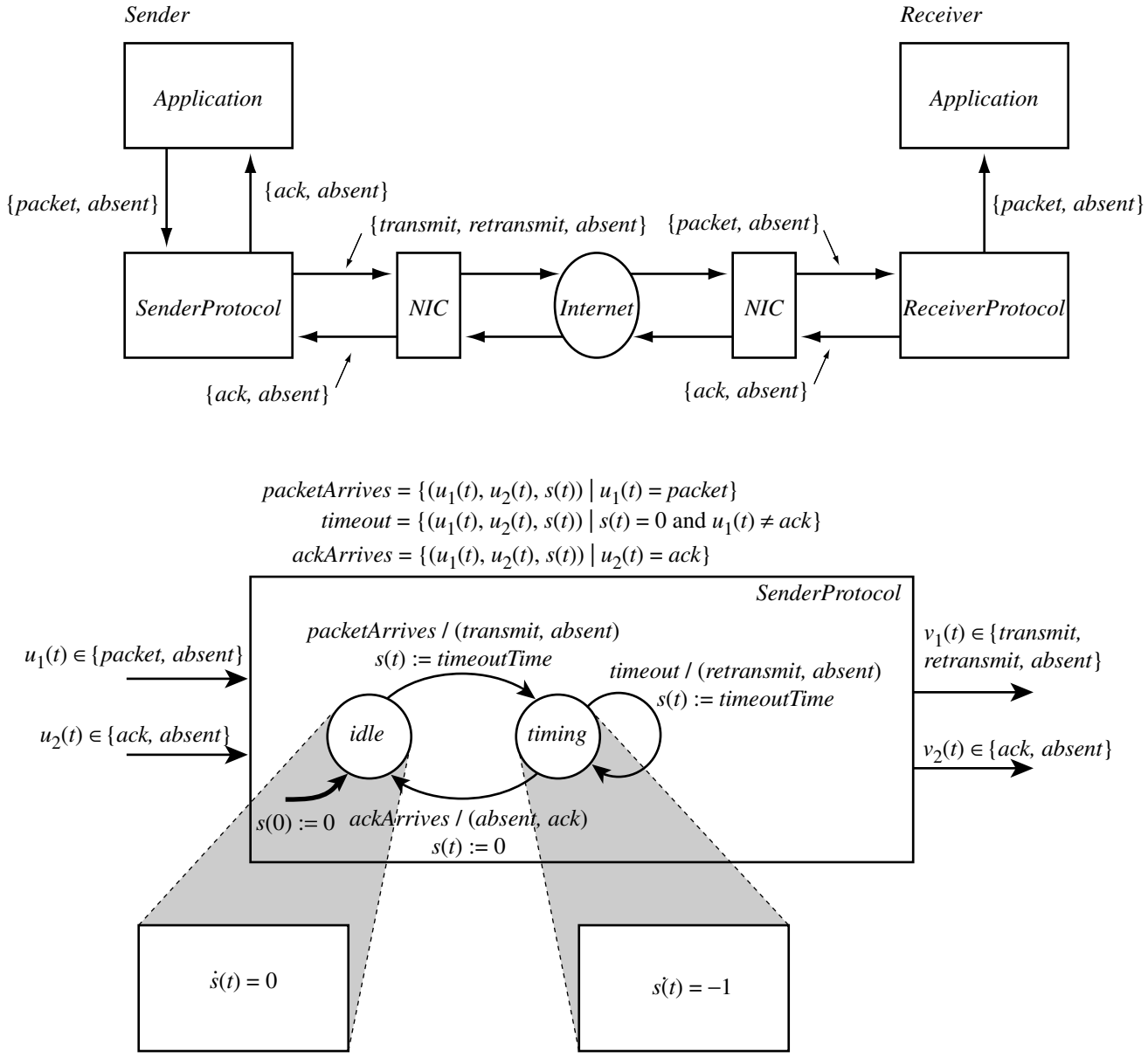


Figure 6.8: The top diagram describes the structure of a communication system. The lower diagram is the timed automaton that implements the sender protocol.

The receiver's NIC sends the *ack* packet back through the network to the *Sender*. The sender's NIC receives this packet and forwards an *ack* to the *SenderProtocol*. The *SenderProtocol* notifies the application that the packet was indeed delivered. The application can now send the next *packet*, and the cycle is repeated until the entire file is delivered.

In reality, the network may drop the packet so that it is not delivered to the receiver, who therefore will not send the corresponding *ack*. The *SenderProtocol* system is designed to take care of this contingency. It is a timed automaton with two modes, *idle* and *timing*, and one refinement state s corresponding to a clock. Initially it is in the *idle* mode and $s(0) = 0$. In the idle mode, $\dot{s}(t) = 0$, so the refinement state remains at zero. When *SenderProtocol* receives a *packet* it makes a transition to the *timing* mode, sends the output event *transmit* to its NIC, and resets s to a timeout value *timeoutTime*.

In the *timing* mode, there are two possibile transitions. In the normal case, the input event *ack* is received before the guard $\{s(t) == 0\}$ is satisfied. The transition to mode *idle* is taken, the output event *ack* is sent to the application, and the clock value $s(t)$ is reset to 0. The system waits for another *packet* from the application. In the second case, the guard $\{s(t) == 0\}$ is satisfied (before event *ack*), and the self-loop transition is taken. In this case, the output event *retransmit* is sent to the NIC, and $s(t)$ is reset to *timeoutTime*.

Notice a feature of this design that may not be expected. If a packet arrives while the machine is in mode *timing*, the packet is ignored. What happens if a packet happens to arrive simultaneously with an *ack* while the machine is mode *timing*?

Exercise 10 asks you to construct the corresponding receiver protocol, which is simpler.

In summary, the *SenderProtocol* machine repeatedly retransmits a packet every *timeoutTime* seconds until it receives an *ack*. This reveals a flaw in the protocol. If the network is for some reason unable ever to successfully transmit a packet to the receiver, the machine will continue retransmission for ever. A better protocol would retransmit a packet a certain number of times, say five times, and if it is unsuccessful, it would return to *idle* and send a message *connectionFailed* to the application. The hybrid system of figure 6.9 incorporates this feature by adding another clock whose value is $r(t)$.

6.4 More interesting dynamics

In timed automata, all that happens in the time-based refinement systems is that time passes. Hybrid systems, however, are much more interesting when the behavior of the refinements is more complex.

Example 6.8: Consider the physical system depicted in figure 6.11. Two sticky round masses are attached to springs. The springs are compressed or extended and then released. The masses oscillate on a frictionless table. If they collide, they stick together and oscillate together. After some time, the stickiness decays, and masses pull apart again.

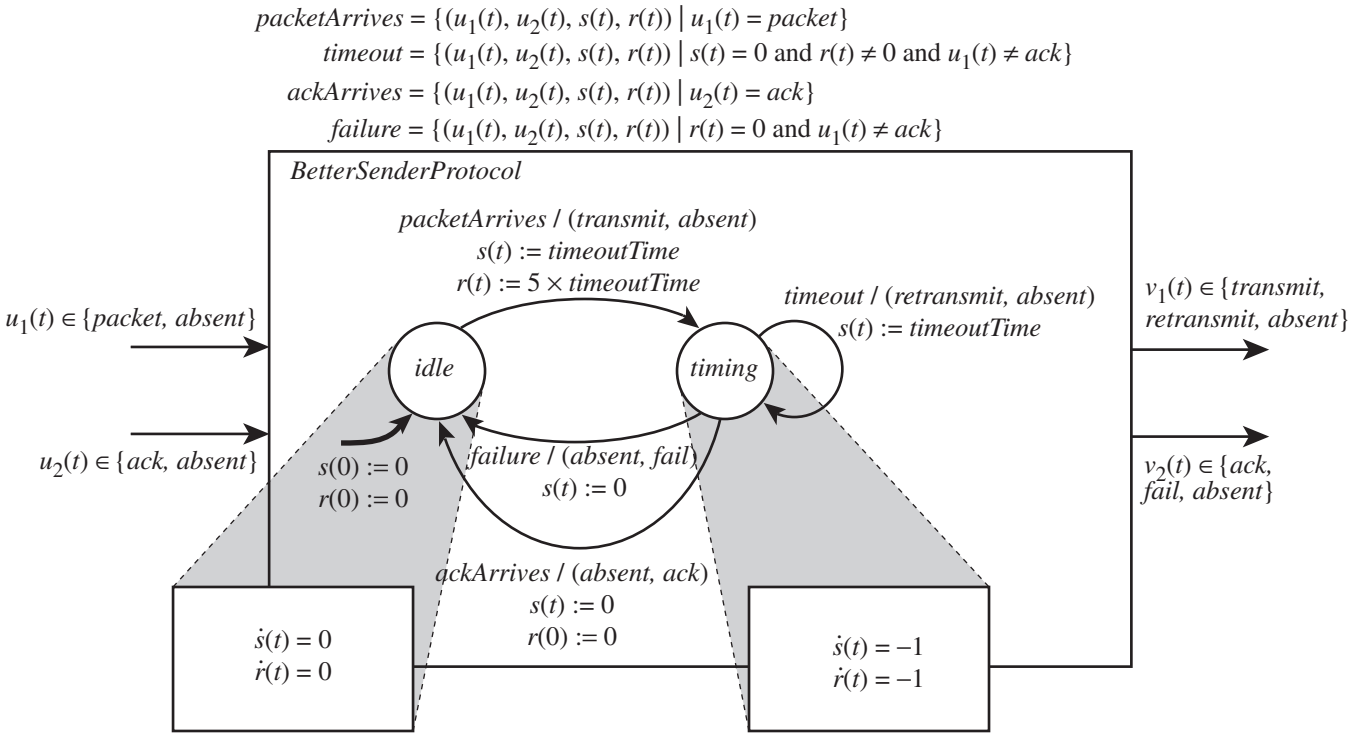


Figure 6.9: An improved sender protocol with two clocks, one of which detects a failed connection.

Probing further: Internet protocols

Communication between two computers, *Sender* and *Receiver*, each connected to the internet, is coordinated by a set of **protocols**. Each protocol can be modeled by a pair of hybrid systems, one in the *Sender* and the other in the *Receiver*. These protocols are arranged in a **protocol stack**, as shown in figure 6.10. Each layer in the stack performs a certain function, and interacts with the corresponding layer in the other computer. The physical layer converts a bit stream into an electrical signal and vice versa and transfers the signal over one link of the network.

The network itself consists of many physical links connected by routers. The routers themselves act as computers, but are missing the higher levels of the protocol stack. The physical layer transports bits over wires, optical fibers, or radio links. The **medium access layer** manages contention for the physical communication resource, preventing collisions among multiple users of the link. The **network layer** routes packets appropriately through the network. The **transport layer** ensures that the end-to-end transfer of packets is reliable, even if the network layer service is unreliable. The **application layer** converts whatever information is to be sent (such as an image or e-mail) into packets and then reassembles the packets into the appropriate information.

This layered approach provides an *abstraction* mechanism. Each layer conceptually interacts with the corresponding layer at a remote machine, as suggested by the dotted lines in figure 6.10. Each layer provides a "service" to the layer above it, using the service offered by the layer underneath. For example, the medium access layer offers as a service the transfer of a packet over a single link. The network layer uses this service to transfer a packet over a sequence of links between the end hosts. This abstraction mechanism permits the design of a single layer, say the transport layer, assuming the service of the network layer, without regard to the layers below the network layer. The hybrid system in example 6.7, for instance, models only the transport layer.

The transport layer in an end-to-end protocol, so it is implemented only at the end points in the connection, as shown in figure 6.10. The routers in the network only need to implement the lower layers.

As mentioned, each protocol layer is modeled as a pair of hybrid systems. Typically, these are timed automata, since coordination between end hosts is achieved via several clocks as in example 6.7. When a guard associated with a clock is satisfied, this signals some contingency in the communication, just as the timeout of the clock in figure 6.8 signals that a packet may be lost.

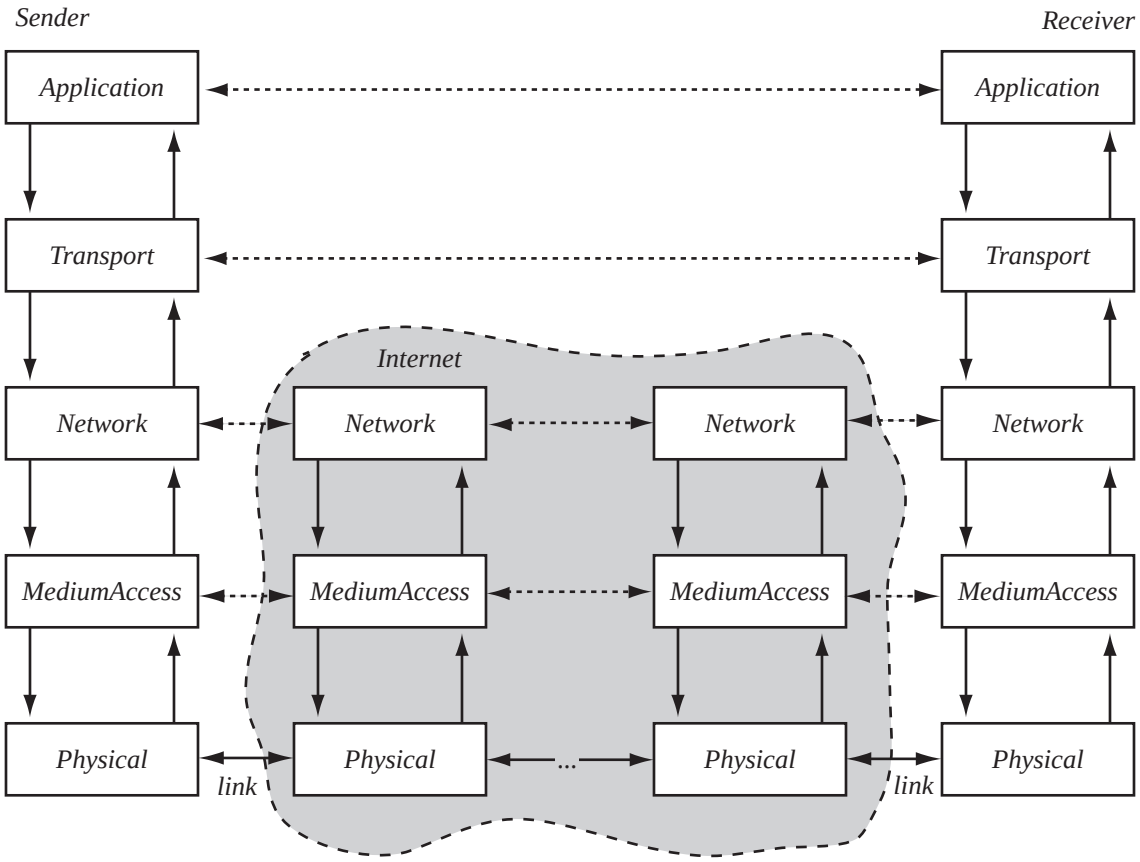


Figure 6.10: Network protocols are organized in a stack. Each protocol interacts with the corresponding layer in a remote computer. The dotted lines indicate conceptual interactions, whereas the solid lines indicate physical interactions.

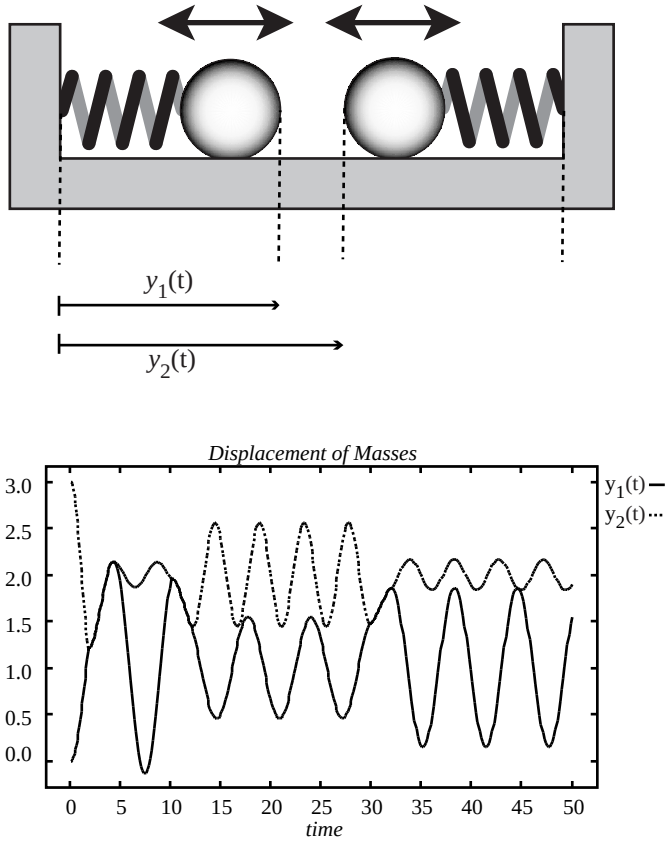


Figure 6.11: Sticky masses system considered in example 6.8.

A plot of the displacement of the two masses as a function of time is shown in the figure. Both springs begin compressed, so the masses begin moving towards one another. They almost immediately collide, and then oscillate together for a brief period until they pull apart. In this plot, they collide two more times, and almost collide a third time.

The physics of this problem is quite simple if we assume idealized springs. Let $y_1(t)$ denote the right edge of the left mass at time t, and $y_2(t)$ denote the left edge of the right mass at time t, as shown in figure 6.11. Let p_1 and p_2 denote the neutral positions of the two masses, i.e. when the springs are neither extended nor compressed, so the force is zero. For an ideal spring, the force at time t on the mass is proportional to $p_1 - y_1(t)$ (for the left mass) and $p_2 - y_2(t)$ (for the right mass). The force is positive to the right and negative to the left.

Let the spring constants be k_1 and k_2, respectively. Then the force on the left spring is $k_1(p_1 - y_1(t))$, and the force on the left spring is $k_2(p_2 - y_2(t))$. Let the masses be m_1 and m_2 respectively. Now we can use Newton's law, which relates force, mass, and acceleration,

$$f = ma.$$

The acceleration is the second derivative of the position with respect to time, which we write $\ddot{y_1}(t)$ and $\ddot{y_2}(t)$ respectively. Thus, as long as the masses are separate, their dynamics are given by

$$\ddot{y_1}(t) = k_1(p_1 - y_1(t))/m_1 \tag{6.1}$$
$$\ddot{y_2}(t) = k_2(p_2 - y_2(t))/m_2. \tag{6.2}$$

When the masses collide, however, the situation changes. With the masses stuck together, they behave as a single object with mass $m_1 + m_2$. This single object is pulled in opposite directions by two springs. While the masses are stuck together, $y_1(t) = y_2(t)$. Let

$$y(t) = y_1(t) = y_2(t).$$

The dynamics are then given by

$$\ddot{y}(t) = \frac{k_1 p_1 + k_2 p_2 - (k_1 + k_2) y(t)}{m_1 + m_2}. \tag{6.3}$$

It is easy to see now how to construct a hybrid systems model for this physical system. The model is shown in figure 6.12. It has two modes, *apart* and *together*. The refinement of the *apart* mode is given by (6.1) and (6.2), while the refinement of the *together* mode is given by (6.3).

We still have work to do, however, to label the transitions. The initial transition is shown in figure 6.12 entering the *apart* mode. Thus, we are assuming the masses begin apart. Moreover, this transition is labeled with an action that sets the initial refinement state. Intuitively, the initial state of the masses is their positions and their initial velocities. In fact, we can define the refinement state to be

$$s(t) = \begin{bmatrix} y_1(t) \\ \dot{y_1}(t) \\ y_2(t) \\ \dot{y_2}(t) \end{bmatrix}.$$

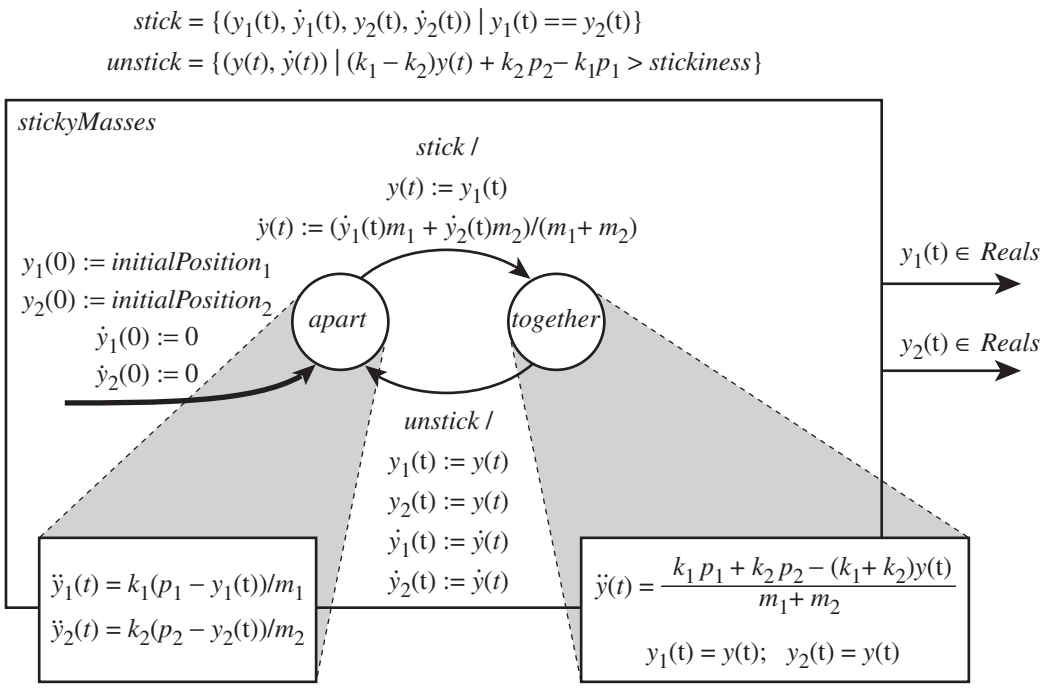


Figure 6.12: Hybrid system model for the sticky masses system considered in example 6.8.

It is then a simple matter to rewrite (6.1) and (6.2) in the form

$$\dot{s}(t) = g(s(t)) \tag{6.4}$$

for a suitably chosen function g (see exercise 12).

In figure 6.12, the initial state has the masses at some specified displacement, and the velocities at zero.

The transition from *apart* to *together* has the guard

$$stick = \{(y_1(t), \dot{y_1}(t), y_2(t), \dot{y_2}(t)) \mid y_1(t) == y_2(t)\}.$$

Thus, when the refinement state of *apart* satisfies this guard, the transition will be taken. No event output is produced, as indicated by the blank after the slash. However, an action is taken to set the initial refinement state of *together*. The refinement state of *together* could be the same $s(t)$ as above, with the additional constraint that $y_1(t) = y_2(t)$ and $\dot{y}_1(t) = \dot{y}_2(t)$, because the masses are stuck together. Or more simply, we could define the state $z(t)$ of *together* to be the position $y(t)$ and velocity $\dot{y}(t)$, where $y(t) = y_1(t) = y_2(t)$,

$$z(t) = \begin{bmatrix} y(t) \\ \dot{y}(t) \end{bmatrix}.$$

The transition from *apart* to *together* sets $y(t)$ equal to $y_1(t)$ (it could equally well have chosen $y_2(t)$, since these are equal). It sets the velocity to conserve momentum. The momentum of the left mass is $\dot{y_1}(t)m_1$, the momentum of the right mass is $\dot{y_2}(t)m_2$, and the momentum of the combined masses is $\dot{y}(t)(m_1 + m_2)$. To make these equal, it sets

$$\dot{y}(t) = \frac{\dot{y_1}(t)m_1 + \dot{y_2}(t)m_2}{m_1 + m_2}.$$

The transition from *together* to *apart* has the more complicated guard

$$unstick = \{(y(t), \dot{y}(t)) \mid (k_1 - k_2)y(t) + k_2p_2 - k_1p_1 > stickiness\}.$$

This guard is satisfied when the right-pulling force on the right mass exceeds the right-pulling force on the left mass by more than the stickiness. The right-pulling force on the right mass is simply

$$f_2(t) = k_2(p_2 - y(t))$$

and the right-pulling force on the left mass is

$$f_1(t) = k_1(p_1 - y(t)).$$

Thus,

$$f_2(t) - f_1(t) = (k_1 - k_2)y(t) + k_2p_2 - k_1p_1.$$

When this exceeds the stickiness, then the masses pull apart.

An interesting elaboration on this example, considered in problem 13, modifies the *together* mode so that the stickiness is initialized to a starting value, but then decays according to the differential equation

$$\dot{s}(t) = -as(t)$$

where $s(t)$ is the stickiness at time t, and a is some positive constant. In fact, it is the dynamics of such an elaboration that is plotted in figure 6.11.

As in example 6.7, it is sometimes useful to have hybrid system models with only one state. The actions on one or more state transitions define the discrete event behavior that combines with the time-based behavior.

Example 6.9: Consider a bouncing ball. At time $t = 0$, the ball is dropped from a height $y(0) = \mathit{initialHeight}$ meters. It falls freely. At some later time t_1 it hits the ground with a velocity $\dot{y}(t_1) < 0$ m/sec. A *bump* event is produced when the ball hits the ground. The collision is inelastic, and the ball bounces back up with velocity $-a\dot{y}(t_1)$, where a is constant in $(0,1)$. The ball will then rise to a certain height and fall back to the ground repeatedly.

The behavior of the bouncing ball can be described by the hybrid system of figure 6.13. There is only one mode, called *free*. When it is not in contact with the ground, we know that the ball follows the second-order differential equation,

$$\ddot{y}(t) = -g, \tag{6.5}$$

where $g = 10$ m/sec^2 is the acceleration imposed by gravity. We can define the refinement state of the *free* mode to be

$$s(t) = \begin{bmatrix} y(t) \\ \dot{y}(t) \end{bmatrix}$$

with the initial conditions $y(0) = \mathit{initialHeight}$ and $\dot{y}(0) = 0$. It is then a simple matter to rewrite (6.5) as a first-order differential equation,

$$\dot{s}(t) = f(s(t)) \tag{6.6}$$

for a suitably chosen function f (see exercise 12).

At the time t_1 when the ball first hits the ground, the guard

$$\mathit{hit} = \{(y(t), \dot{y}(t)) \mid y(t) = 0\}$$

is satisfied, and the self-loop transition is taken. The output *bump* is produced, and the action $\dot{y}(t) := -a\dot{y}(t)$ assigns $\dot{y}(t_1+) = -a\dot{y}(t_1)$. Here, $\dot{y}(t_1+)$ is the velocity after the bump, and $\dot{y}(t_1)$ is the velocity before the bump. Then (6.5) is followed again until the guard becomes true again.

By integrating (6.5) we get, for all $t \in (0, t_1)$,

$$\begin{aligned} \dot{y}(t) &= -gt, \\ y(t) &= y(0) + \int_0^t \dot{y}(\tau)d\tau = \mathit{initialHeight} - \frac{1}{2}gt^2. \end{aligned}$$

So $t_1 > 0$ is determined by $y(t_1) = 0$. It is the solution to the equation

$$\mathit{initialHeight} - \frac{1}{2}gt^2 = 0.$$

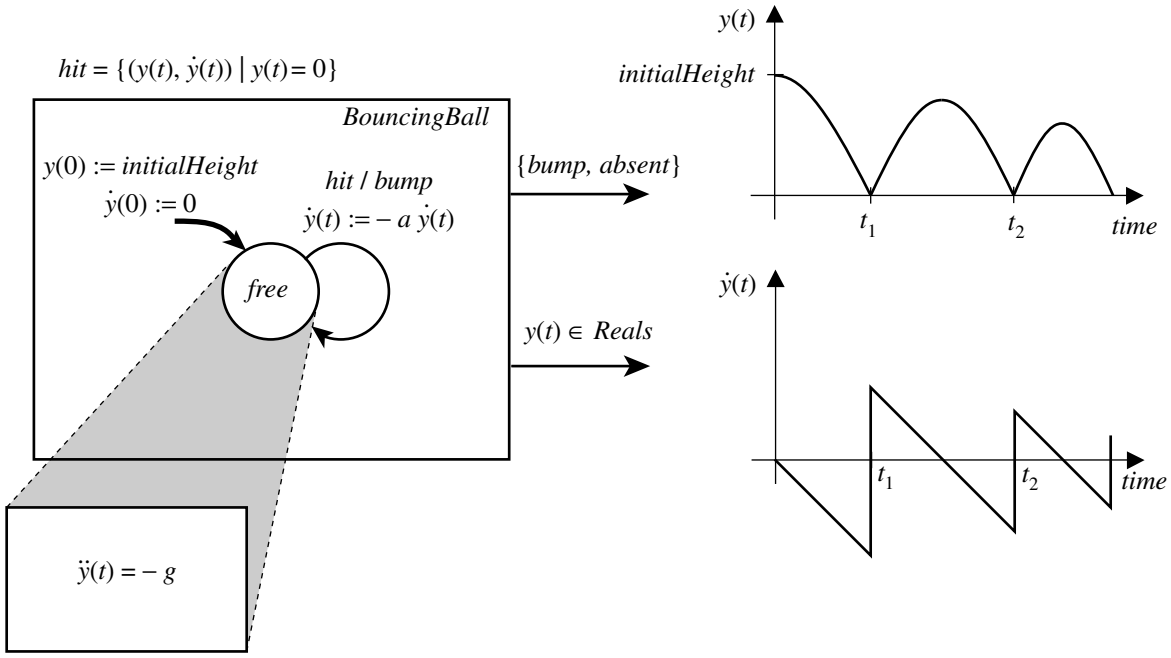


Figure 6.13: The motion of a bouncing ball may be described as a hybrid system with only one mode. The system outputs a *bump* each time the ball hits the ground, and also outputs the position of the ball. The position and velocity are plotted versus time at the right.

Thus,

$$t_1 = \sqrt{2\ initialHeight/g}.$$

Figure 6.13 plots the refinement state versus time.

6.5 Supervisory control

We introduce supervisory control through a detailed example. A control system involves four components. There is a system called the **plant**—the physical process that is to be controlled; the environment in which the plant operates; the sensors that measure some variables of the plant and the environment; and the controller that determines the mode transition structure and selects the time-based inputs to the plant. The controller has two levels: the supervisory control that determines the mode transition structure, and the 'low-level' control that selects the time-based inputs which control the behavior of the refinements. A complete design includes both levels of control as in the following example.

Example 6.10: The plant is an **automated guided vehicle** or **AGV** that moves along a closed track painted on a warehouse or factory floor. We will design a controller so

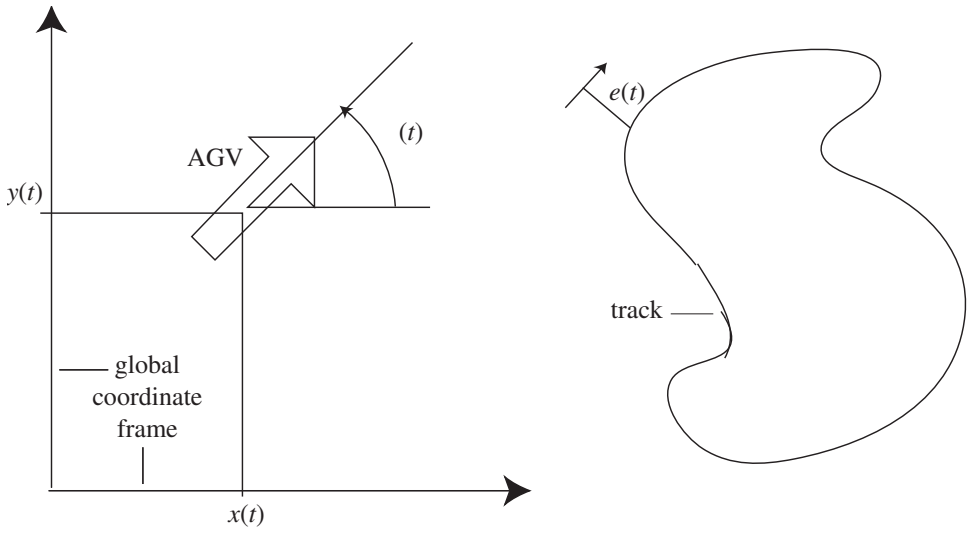


Figure 6.14: Illustration of the automated guided vehicle of example 6.10. The vehicle is shown as a large arrow on the left and as a small arrow on the right. On the right, the vehicle is following a curved painted track, and has deviated from the track by a distance $e(t)$. The coordinates of the vehicle at time t with respect to the global coordinate frame are $(x(t), y(t), \phi(t))$.

that the vehicle closely follows the track.

The vehicle has two degrees of freedom. At any time t, it can move forward along its body axis with speed $u(t)$ with the restriction that $0 \leq u(t) \leq 10$ mph. It can also rotate about its center of gravity with an angular speed $\omega(t)$ restricted to $-\pi \leq \omega(t) \leq \pi$ radians/second. We ignore the inertia of the vehicle.

Let $(x(t), y(t)) \in \mathit{Reals}^2$ be the position and $\phi(t) \in [-\pi, \pi]$ the angle (in radians) of the vehicle at time t relative to some fixed coordinate frame, as shown on the left in figure 6.14. In terms of this coordinate frame, the motion of the vehicle is given by a system of three differential equations,

$$\begin{aligned} \dot{x}(t) &= u(t)\cos\phi(t), \\ \dot{y}(t) &= u(t)\sin\phi(t), \\ \dot{\phi}(t) &= \omega(t). \end{aligned} \tag{6.7}$$

The track and the vehicle are shown on the right of figure 6.14. Equations (6.7) describe the plant. The environment is the closed painted track. It could be described by an equation. We will describe it indirectly below by means of a sensor.

The two-level controller design is based on a simple idea. The vehicle always moves at its maximum speed of 10 mph. If the vehicle strays too far to the left of the track, the controller steers it towards the right; if it strays too far to the right of the track, the controller steers it towards the left. If the vehicle is close to the track, the controller

maintains the vehicle in a straight direction. Thus the controller guides the vehicle in four modes, *left*, *right*, *straight*, and *stop*. In *stop* mode an operator may bring the vehicle to a halt.

The following differential equations govern the AGV's motion in the refinement of the four modes. They describe the low-level controller, i.e. the selection of the time-based inputs in each mode.

straight

$$\begin{aligned} \dot{x}(t) &= 10\cos\phi(t) \\ \dot{y}(t) &= 10\sin\phi(t) \\ \dot{\phi}(t) &= 0 \end{aligned}$$

left

$$\begin{aligned} \dot{x}(t) &= 10\cos\phi(t) \\ \dot{y}(t) &= 10\sin\phi(t) \\ \dot{\phi}(t) &= \pi \end{aligned}$$

right

$$\begin{aligned} \dot{x}(t) &= 10\cos\phi(t) \\ \dot{y}(t) &= 10\sin\phi(t) \\ \dot{\phi}(t) &= -\pi \end{aligned}$$

stop

$$\begin{aligned} \dot{x}(t) &= 0 \\ \dot{y}(t) &= 0 \\ \dot{\phi}(t) &= 0 \end{aligned}$$

In the *stop* mode, the vehicle is stopped, $x(t), y(t), \phi(t)$ are constant. In the *left* mode, $\phi(t)$ increases at the rate of π radians/second, so from figure 6.14 we see that the vehicle moves to the left. In the *right* mode, it moves to the right. In the *straight* mode, $\phi(t)$ is constant, and the vehicle moves straight ahead with a constant heading. The refinements of the four modes are shown in the boxes of figure 6.15.

We design the supervisory control governing transitions between modes in such a way that the vehicle closely follows the track, using a sensor that determines how far the vehicle is to the left or right of the track. We can build such a sensor using photodiodes. Let's suppose the track is painted with a light-reflecting color, whereas the floor is relatively dark. Underneath the AGV we place an array of photodiodes as shown in figure 6.16. The array is perpendicular to the AGV body axis. As the AGV passes over the track, the diode directly above the track generates more current than the other diodes. By comparing the magnitudes of the currents through the different diodes, the sensor gives the displacement $e(t)$ of the center of the array (hence, the center of the

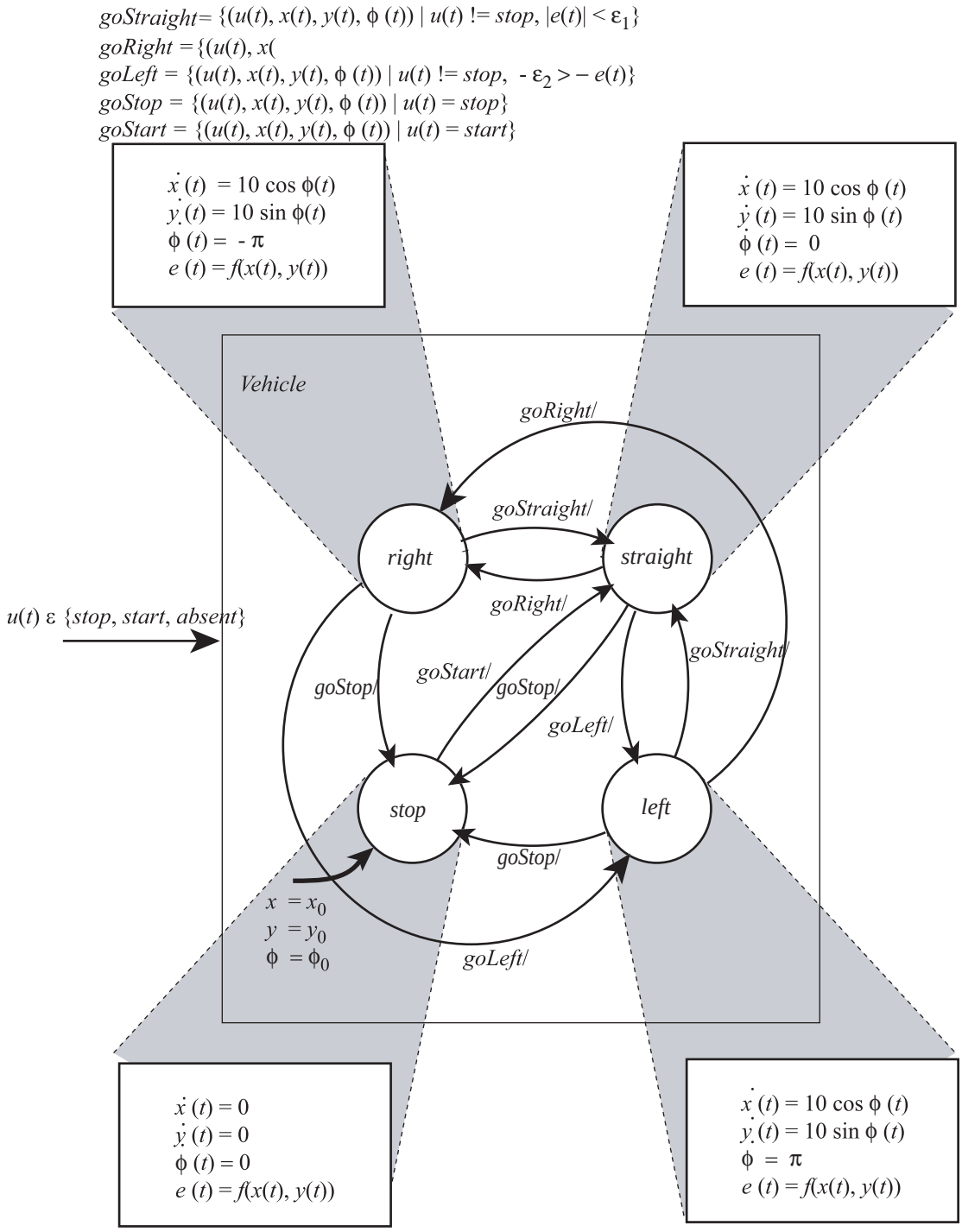


Figure 6.15: The automatic guided vehicle of example 6.10 has four modes: *stop*, *straight*, *left*, *right*.

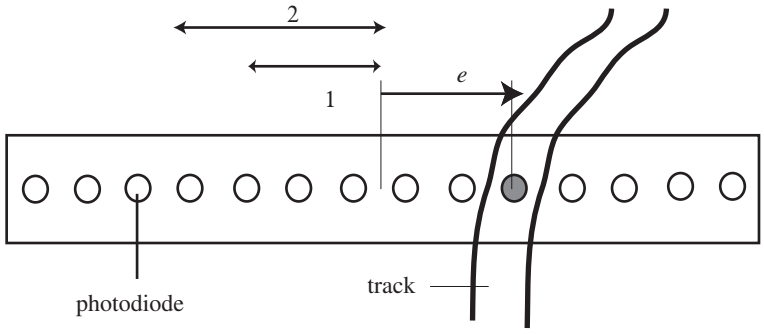


Figure 6.16: An array of photodiodes under the AGV is used to estimate the displacement e of the AGV relative to the track. The photodiode directly above the track generates more current.

AGV) from the track. We adopt the convention that $e(t) < 0$ means that the AGV is to the right of the track and $e(t) > 0$ means it is to the left. We model the sensor output as a function f of the AGV's position,

$$\forall t, \quad e(t) = f(x(t), y(t)).$$

The function f of course depends on the environment—the track. We now specify the supervisory controller precisely. We select two thresholds, $0 < \varepsilon_1 < \varepsilon_2$, as shown in figure 6.16. If the magnitude of the displacement is small, $|e(t)| < \varepsilon_1$, we consider that the AGV is close enough to the track, and the AGV can move straight ahead, in *straight* mode. If $0 < \varepsilon_2 < e(t)$ ($e(t)$ is large and positive), the AGV has strayed too far to the left and must be steered to the right, by switching to *right* mode. If $0 > -\varepsilon_2 > e(t)$ ($e(t)$ is large and negative), the AGV has strayed too far to the right and must be steered to the left, by switching to *left* mode. This control logic is captured in the mode transitions of figure 6.15. The input events are $\{stop, start, absent\}$. By selecting events *stop* and *start* an operator can stop or start the AGV. There is no time-based input. There is no external output. The initial mode is *stop*, and the initial values of its refinement are (x_0, y_0, ϕ_0).

We analyze how the AGV will move. Figure 6.17 sketches one possible trajectory. Initially the vehicle is within distance ε_1 of the track, so it moves straight. At some later time, the vehicle goes too far to the left, the guard

$$goRight = \{(u(t), x(t), y(t), \phi(t)) \mid u(t) \neq stop, \varepsilon_2 < e(t)\}$$

is satisfied, and there is a mode switch to *right*. After some time, the vehicle is close enough to the track, the guard

$$goStraight = \{(u(t), x(t), y(t), \phi(t)) \mid u(t) \neq stop, |e(t)| < \varepsilon_1\}$$

is satisfied, and there is a mode switch to *straight*. Some time later, the vehicle is too far to the right, the guard

$$goLeft = \{(u(t), x(t), \phi(t)) \mid u(t) \neq stop \mid -\varepsilon_2 > e(t)\}$$

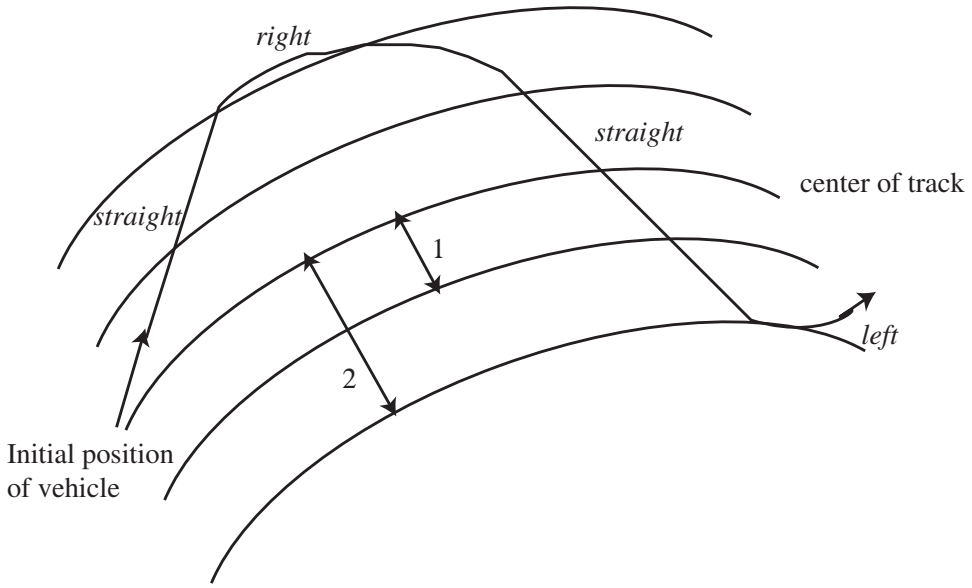


Figure 6.17: A trajectory of the AGV, annotated with modes.

is satisfied, there is a mode switch to *left*, and so on.

The example illustrates the four components of a control system. The plant is described by the differential equations (6.7) that govern the evolution of the refinement state at time t, $(x(t), y(t), \phi(t))$, in terms of the time-based input, $(u(t), \omega(t))$. The second component is the environment—the closed track. The third component is the sensor, whose output at time t, $e(t) = f(x(t), y(t))$, gives the position of the AGV relative to the track. The fourth component is the two-level controller. The supervisory controller comprises the four modes and the guards that determine when to switch between modes. The low-level controller specifies how the time-based inputs, u and ω, are selected in each mode.

6.6 Formal model

We develop a formal model of a hybrid system similar to the 'sets and functions' model of section 3.1. A hybrid system *HybridSystem* is a 5-tuple,

$$HybridSystem = (States, Inputs, Outputs, TransitionStructure, initalState),$$

where, *States*, *Inputs*, *Outputs* are sets, and $initalState \in States$ is the initial state. *TransitionStructure* consists of several items that determine how the hybrid system evolves in time $t \in T$. T may be $Reals_+$ or $Naturals_0$. Here we assume $T = Reals_+$.

$States = Modes \times RefinementStates$ is the state space. *Modes* is the finite set of modes. *RefinementStates* is the state space of the refinements. If the current state at time t is $(m(t), s(t))$ we say that the system is in mode $m(t)$ and its refinement is in state $s(t)$.

$Inputs = InputEvents \times TimeBasedInputs$ is the set of input symbols. *InputEvents* is the finite alphabet of discrete input symbols, including a stuttering symbol, while *TimeBasedInputs* is the set of input values to which the refinement reacts. An input signal consists of a pair of functions (u,x) where u:$Reals_+ \rightarrow InputEvents$ and x:$Reals_+ \rightarrow TimeBasedInputs$. For all except a discrete set of times t, $u(t)$ is the stuttering symbol, *absent*.

$Outputs = OutputEvents \times TimeBasedOutputs$ is the set of output symbols. *OutputEvents* is the finite alphabet of discrete output symbols, including a stuttering output, *absent*, and *TimeBasedOutputs* is the set of continuous output values. An output signal consists of a pair of functions (v,y) where v:$Reals_+ \rightarrow OutputEvents$ and y:$Reals_+ \rightarrow TimeBasedOutputs$. For all except a discrete set of times, $v(t) = absent$.

The transition structure determines how a mode transition occurs and how the refinement state changes over time. Suppose the inputs signal is (u,x). Suppose at time t the mode is m and the refinement state is s. For each destination mode d there is a guard

$$\begin{aligned} G_{m,d} &= U_{m,d} \times X_{m,d} \times S_{m,d} \\ &\subset InputEvents \times TimeBasedInputs \times RefinementStates. \end{aligned}$$

There is also an output event, say $v_{m,d}$, and an action $A_{m,d}$:$RefinmentStates \rightarrow RefinmentStates$ that assigns a (possibly new) value to each refinement state, (possibly) depending on the current value of the refinement state. If there is a match $(u(t),x(t),s(t)) \in G_{m,d}$, then there is a discrete transition at t; the mode after the transition is d, the output event $v(t) = v_{m,d}$ is produced, and the refinement state in mode d at time $t+$ immediately after the transition is set to $s(t+) = A_{m,d}(s(t))$.

If no guard is satisfied at time t, then the refinement state $s(t)$ and the time-based output $y(t)$ are determined by the time-based input signal x according to the equations governing the refinement dynamics. Here we will need to be concrete. In all of the examples above, we have taken

$$\begin{aligned} RefinementStates &= Reals^N, \\ TimeBasedInputs &= Reals^M, \text{ and} \\ TimeBasedOutputs &= Reals^K. \end{aligned}$$

In this concrete setting, the refinement dynamics are given as

$$\forall t \in T_m, \quad \dot{s}(t) = f_m(s(t),x(t)), \tag{6.8}$$

$$y(t) = g_m(s(t),x(t), \tag{6.9}$$

where $T_m \subset T$ is the set of times t when the system is in mode m, and the functions

$$\begin{aligned} f_m : Reals^N \times Reals^M: &\rightarrow Reals^N, \\ g_m : Reals^N \times Reals^M: &\rightarrow Reals^K \end{aligned}$$

characterize the behavior of the refinement system in mode m. The function

$$s : Reals_+ \rightarrow RefinementStates$$

is the trajectory of the refinment states.

We can now see how the hybrid system evolves over time. At time $t = 0$, the system starts in the initial state, say $(m(0), s(0))$. It evolves in alternating phases of time passage, $(t_0 = 0, t_1], (t_1, t_2], \cdots$, and discrete transitions at $t_1, t_2, \cdots$. During the first interval $(t_0, t_1]$, no guard is satisfied and the system remains in mode $m(0)$; the refinement state $s(t)$ and time-based output $y(t)$ are determined by (6.8), (6.9); and the discrete event output $v(t) = absent$.

At time t_1, the guard $G_{m(0),m(1)}$ for some destination mode $m(1)$ is matched by $(u(t_1), x(t_1), s(t_1))$. There is a mode transition to $m(1)$, the output event $v(t_1)$ is produced, and the continuous state is set to $s(t_1+) = A_{m(0)m(1)}(s(t_1))$. The discrete transition phase is now over, and the system begins the time passage phase in the new mode $m(1)$ and the continuous state $s(t_1+)$.

6.7 Summary

Hybrid systems provide a bridge between time-based models and state-machine models. The combination of the two families of models provides a rich framework for describing real-world systems. There are two key ideas. First, discrete events are embedded in a time base. Second, a hierarchical description is particularly useful, where the system undergoes discrete transitions between different modes of operation. Associated with each mode of operation is a time-based system called the refinement of the mode. Mode transitions are taken when guards that specify the combination of inputs and refinement states are satisfied. The action associated with a transition, in turn, sets the refinement state in the destination mode.

The behavior of a hybrid system is understood using the tools of state machine analysis for mode transitions the tools of time-based analysis for the refinement systems. The design of hybrid systems similarly proceeds on two levels: state machines are designed to achieve the appropriate logic of mode transitions, and refinement systems are designed to secure the desired time-based behavior in each mode.

Exercises

In some of the following exercises you are asked to design state machines that carry out a given task. The design is simple and elegant if the state space is properly chosen. Although the state space is not unique, there often is a natural choice. As usual, each problem is annotated with the letter **E, T, C** which stands for exercise, requires some thought, requires some conceptualization. Problems labeled **E** are usually mechanical, those labeled **T** require a plan of attack, those labeled **C** usually have more than one defensible answer.

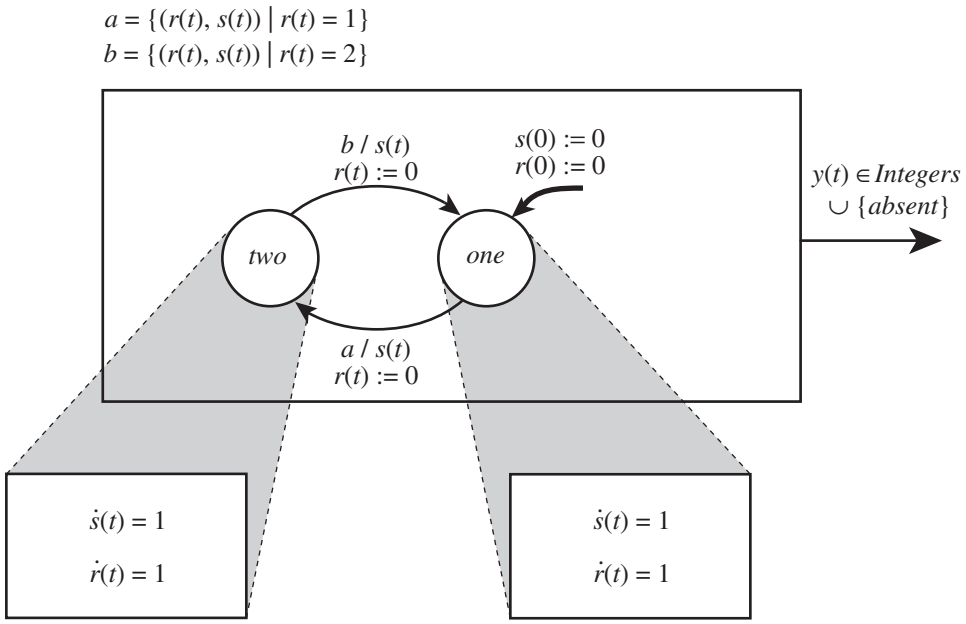
1. **C** Consider the loudness compensation of example 6.3. Suppose that instead of a switch on the front panel, the system automatically selects from among four compensation filters with state-space models $[A, b, c_1, d_1]$, $[A, b, c_2, d_2]$, $[A, b, c_3, d_3]$, and $[A, b, c_4, d_4]$. The A matrix and b vector are the same for all four. Which filter is used depends on a discrete-time v input, where at index n, $v(n)$ represents the current volume level. When the volume is high, above

some threshold, filter 4 should be used. When it is low, filter 1 should be used. Design a hybrid system that does this.

2. **E** Construct a timed automaton similar to that of figure 6.4 which produces *tick* at times $1, 2, 3, 5, 6, 7, 8, 10, 11, \cdots$. That is, ticks are produced with intervals between them of 1 second (three times) and 2 seconds (once).

3. **E** The objective of this problem is to understand a timed automaton, and then to modify it as specified.

   (a) For the timed automaton shown below, describe the output y. Avoid imprecise or sloppy notation.



   (b) Assume there is a new input u: *Reals* $\rightarrow$ *Inputs* with alphabet

$$Inputs = \{reset, absent\},$$

   and that when the input has value *reset*, the hybrid system starts over, behaving as if it were starting at time 0 again. Modify the hybrid system from part (a) so that it behaves like the system in (a).

4. **E** You have an analog source that produces a pure tone. You can switch the source on or off by the input event *on* or *off*. Construct a system that upon receiving an input event *ring* produces an 80 ms-long sound consisting of three 20 ms-long bursts of the pure tone separated by two 10 ms intervals of silence. What does your system do if it receives two *ring* events that are 50 ms apart?

5. **C** Automobiles today have the features listed below. Implement each feature as a timed automaton.

(a) The dome light is turned on as soon as any door is opened. It stays on for 30 seconds after all doors are shut. What sensors are needed?

(b) Once the engine is started, a beeper is sounded and a red light warning is indicated if there are passengers that have not buckled their seat belt. The beeper stops sounding after 30 seconds, or as soon the seat belts are buckled, whichever is sooner. The warning light is on all the time the seat belt is unbuckled. **Hint:** Assume the sensors provide a *warn* event when the ignition is turned on and there is a seat with passenger not buckled in, or if the ignition is already on and a passenger sits in a seat without buckling the seatbelt. Assume further that the sensors provide a *noWarn* event when a passenger departs from a seat, or when the buckle is buckled, or when the ignition is turned off.

6. **E** A programmable thermostat allows you to select 4 times, $0 \leq T_1 \leq \cdots \leq T_4 < 24$ (for a 24-hour cycle) and the corresponding temperatures $a_1, \cdots, a_4$. Construct a timed automaton that sends the event a_i to the heating systems controller. The controller maintains the temperature close to the value a_i until it receives the next event. How many timers and modes do you need?

7. **E** Construct a parking meter similar to that in figure 6.6 that allows a maximum of 30 minutes (rather than 60 minutes) and accepts *coin5* and *coin25* as inputs. Then draw the state trajectories (both the mode and the clock state) and the output signal when *coin5* occurs at time 0, *coin25* occurs at time 3, and then there is no input event for the next 35 minutes.

8. **T** Consider the timed automaton of figure 6.6. Suppose we view the box as a discrete-event system with input alphabet $\{coin5, coin25, absent\}$ and output alphabet $\{expired, absent\}$. Does the box behave as a finite state machine?

9. **C** Figure 6.18 depicts the intersection of two one-way streets, called Main and Secondary. A light on each street controls its traffic. Each light goes through a cycle consisting of a red (R), green (G), and yellow (Y) phases. It is a safety requirement that when one light is in its green or yellow phase, the other is in its red phase. The yellow phase is always 20 seconds long.

   The traffic lights operate as follows, in one of two modes. In the normal mode, there is a 5 minute-long cycle with the main light having 4 minutes of green and 20 seconds of yellow– the secondary light is red for these 4 minutes and 20 seconds—and 40 seconds of red—during which the secondary light is green for 20 seconds followed by 20 seconds of yellow.

   The second, or interrupt mode works as follows. Its purpose is to quickly give a right of way to the secondary road. A sensor in the secondary road detects if a vehicle has crossed it. When this happens, the main light aborts its green phase and immediately switches to its 20 second yellow phase. If the vehicle is detected while the main light is yellow or red, the system continues in its normal mode.

   Design a hybrid system that controls the lights. Let this hybrid system have discrete outputs that are pairs GG, GY, GR, etc. where the first letter denotes the color of the main light second letter denotes the color of the secondary light.

10. **T** Design a *ReceiverProtocol* hybrid system that works together with the *SenderProtocol* of example 6.7.

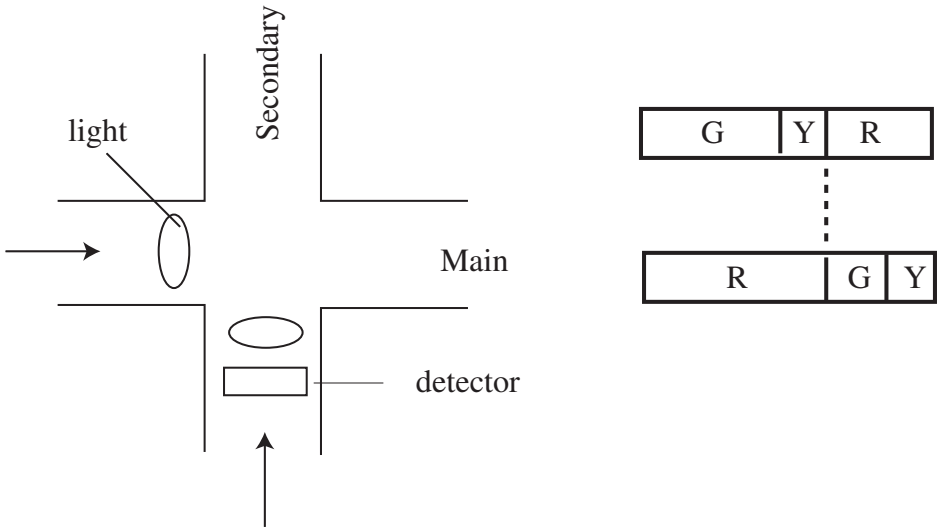


Figure 6.18: Traffic lights control the intersection of a main street and a secondary street. A detector senses when a vehicle crosses it. The red phase of one light must coincide with the green and yellow phases of the other light.

11. **E** For the bouncing ball of example 6.9 let t_n be the time when the ball hits the ground for the n-th time, and let $v(n) = \dot{y}(t_n)$ be the velocity at that time.

    (a) Find a relation between $v(n+1)$ and $v(n)$ and then calculate $v(n)$ in terms of $v(1)$.

    (b) Obtain t_n in terms of $v(n)$.

    (c) Calculate the maximum height reached by the ball after successive bumps.

12. **E** Translate refinement systems that are described as second-order differential equations into first-order differential equations. Specifically:

    (a) For the sticky masses system in example 6.8, find the function g such that (6.1) and (6.2) are represented as (6.4). Is this function linear?

    (b) For the bouncing ball system in example 6.9, find the function f such that (6.5) is represented as (6.6). Is this function linear?

13. **T** Elaborate the hybrid system model of figure 6.12 so that in the *together* mode, the stickiness decays according to the differential equation

$$\dot{s}(t) = -as(t)$$

    where $s(t)$ is the stickiness at time t, and a is some positive constant. On the transition into this mode, the stickiness should be initialized to some starting stickiness b.

14. **T** Show that the trajectory of the AGV of figure 6.15 while it is in *left* or *right* mode is a circle. What is the radius of this circle, and how long does it take to complete a circle?

15. **E** Express the hybrid system of figure 6.15 in terms of the formal model of section 6.6. That is, identify the sets *Inputs*, *Outputs*, and the *TransitionStructure*.