

Chapter 1

Systems and Models

1.1 INTRODUCTION

As its title suggests, this book is about a special class of *systems* which in recent decades have become an integral part of our world. Before getting into the details of this particular class of systems, it is reasonable to start out by simply describing what we mean by a “system”, and by presenting the fundamental concepts associated with system theory as it developed over the years. This defines the first objective of this chapter, which is for the benefit of readers with little or no prior exposure to introductory material on systems and control theory (Sect. 1.2). Readers who are already familiar with concepts such as “state spaces”, “state equations”, “sample paths”, and “feedback” may immediately proceed to Sect. 1.3.

The second objective is to look at useful classifications of systems so as to reveal the features motivating our study of *discrete event* systems. Historically, scientists and engineers have concentrated on studying and harnessing natural phenomena which are well-modeled by the laws of gravity, classical and nonclassical mechanics, physical chemistry, etc. In so doing, we typically deal with quantities such as the displacement, velocity, and acceleration of particles and rigid bodies, or the pressure, temperature, and flow rates of fluids and gases. These are “continuous variables” in the sense that they can take on any real value as *time* itself “continuously” evolves. Based on this fact, a vast body of mathematical tools and techniques has been developed to model, analyze, and control the systems around us. It is fair to say that the study of ordinary and partial differential equations currently provides the main infrastructure for system analysis and control.

But in the day-to-day life of our technological and increasingly computer-dependent world, we notice two things. First, many of the quantities we deal with are “discrete”, typically involving counting integer numbers (how many parts are in an inventory, how many planes are on a runway, how many telephone calls are active). And second, what

drives many of the processes we use and depend on are instantaneous “events” such as the pushing of a button, hitting a keyboard key, or a traffic light turning green. In fact, much of the technology we have invented and rely on (especially where digital computers are involved) is event-driven: communication networks, manufacturing facilities, or the execution of a computer program are typical examples.

1.2 SYSTEM AND CONTROL BASICS

In this section, we will introduce carefully, but rather informally, the basic concepts of system theory. As we go along, we will identify fundamental criteria by which systems are distinguished and classified. These are summarized in Sect. 1.4, Fig. 1.31. Since this section may be viewed as a summary of system and control engineering basics, it may be skipped by readers with a background in this area, who can immediately proceed to Sect. 1.3 introducing the key elements of discrete event systems. By glancing at Fig. 1.31, these readers can also immediately place the class of discrete event systems in perspective.

1.2.1 The Concept of System

System is one of those primitive concepts (like *set* or *mapping*) whose understanding might best be left to intuition rather than an exact definition. Nonetheless, we can provide three representative definitions found in the literature:

- An aggregation or assemblage of things so combined by nature or man as to form an integral or complex whole (*Encyclopedia Americana*).
- A regularly interacting or interdependent group of items forming a unified whole (*Webster's Dictionary*).
- A combination of components that act together to perform a function not possible with any of the individual parts (*IEEE Standard Dictionary of Electrical and Electronic Terms*).

There are two salient features in these definitions. First, a system consists of interacting “components”, and second a system is associated with a “function” it is presumably intended to perform. It is also worth pointing out that a system should not always be associated with physical objects and natural laws. For example, system theory has provided very convenient frameworks for describing economic mechanisms or modeling human behavior and population dynamics.

1.2.2 The Input–Output Modeling Process

As scientists and engineers, we are primarily concerned with the quantitative analysis of systems, and the development of techniques for design, control, and the explicit measurement of system performance based on well-defined criteria. Therefore, the purely qualitative definitions given above are inadequate. Instead, we seek a *model* of an actual system. Intuitively, we may think of a model as a device that simply duplicates the behavior of the system itself. To be more precise than that, we need to develop some mathematical means for describing this behavior.

To carry out the modeling process, we start out by defining a set of *measurable variables* associated with a given system. For example, particle positions and velocities, or voltages and currents in an electrical circuit, which are all real numbers. By measuring these variables over a period of time $[t_0, t_f]$ we may then collect *data*. Next, we select a subset of these variables and assume that we have the ability to vary them over time. This defines a set of time functions which we shall call the *input variables*

$$\{u_1(t), \dots, u_p(t)\} \quad t_0 \leq t \leq t_f \quad (1.1)$$

Then, we select another set of variables which we assume we can directly measure while varying $u_1(t), \dots, u_p(t)$, and thus define a set of *output variables*

$$\{y_1(t), \dots, y_m(t)\} \quad t_0 \leq t \leq t_f \quad (1.2)$$

These may be thought of as describing the “response” to the “stimulus” provided by the selected input functions. Note that there may well be some variables which have not been associated with either the input or the output; these are sometimes referred to as *suppressed* output variables.

To simplify notation, we represent the input variables through a column vector $\mathbf{u}(t)$ and the output variables through another column vector $\mathbf{y}(t)$; for short, we refer to them as the *input* and *output* respectively. Thus, we will write

$$\mathbf{u}(t) = [u_1(t), \dots, u_p(t)]^T$$

where $[\cdot]^T$ denotes the transpose of a vector, and, similarly,

$$\mathbf{y}(t) = [y_1(t), \dots, y_m(t)]^T$$

To complete a model, it is reasonable to postulate that there exists some mathematical relationship between input and output. Thus, we assume we can define functions

$$y_1(t) = g_1(u_1(t), \dots, u_p(t)), \dots, y_m(t) = g_m(u_1(t), \dots, u_p(t))$$

and obtain the system model in the mathematical form

$$\mathbf{y} = \mathbf{g}(\mathbf{u}) = [g_1(u_1(t), \dots, u_p(t)), \dots, g_m(u_1(t), \dots, u_p(t))]^T \quad (1.3)$$

where $\mathbf{g}(\cdot)$ denotes the column vector whose entries are the functions $g_1(\cdot), \dots, g_m(\cdot)$.

This is the simplest possible modeling process, and it is illustrated in Fig. 1.1. Strictly speaking, a system is “something real” (e.g., an amplifier, a car, a factory, a human body), whereas a model is an “abstraction” (a set of mathematical equations). Often, the model only approximates the true behavior of the system. However, once we are convinced we have obtained a “good” model, this distinction is usually dropped, and the terms *system* and *model* are used interchangeably. This is what we will be doing in the sequel. But, before doing so, it is worth making one final remark. For any given system, it is always possible (in principle) to obtain a model; the converse is not true, since mathematical equations do not always yield real solutions. For example, let $u(t) = -1$ for all t be a scalar input representing a constant voltage, and $y = \sqrt{u} = \sqrt{-1}$ be the output. Clearly, we cannot build any electrical system generating an imaginary voltage. In such cases, we say that a system is not physically *realizable*.

It is important to emphasize the flexibility built into the modeling process, since no unique way to select input and output variables is imposed (see also Example 1.1). Thus, it is the modeler’s task to identify these variables depending on a particular point of view or on the constraints imposed upon us by a particular application.

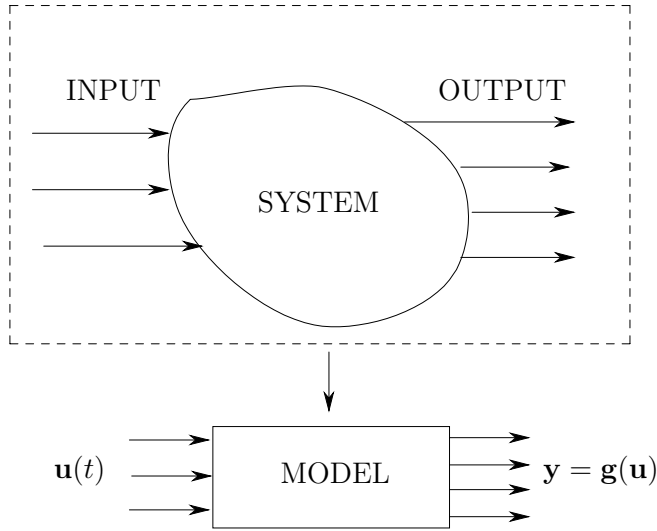


Figure 1.1: Simple modeling process.

Example 1.1 (A voltage-divider system)

The voltage divider circuit of Fig. 1.2 constitutes a simple electrical system. Five variables are shown: the source voltage V , the current i , the two resistances r and R , and the voltage v across R . The simplest models we can construct make use of standard circuit theory relationships such as:

$$v = V \frac{R}{R + r} \quad (1.4)$$

$$v = iR \quad (1.5)$$

Assuming we can adjust the source voltage V (input) and are interested in regulating

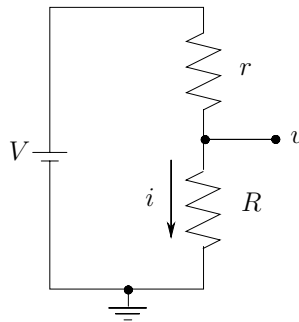


Figure 1.2: Simple electrical system for Example 1.1.

the voltage v (output), we can obtain the model shown in Fig. 1.3(a). Alternatively, suppose the power source is fixed, but we have an adjustable resistance r , in which case our model might be that of Fig. 1.3(b). Finally, if both V and r are adjustable and we are interested in regulating the current i , the model of Fig. 1.3(c) is appropriate.

Thus, for the same underlying system, different models may be conceived. Of course, the functional relationships between variables do not change, but only the choice of input and output.

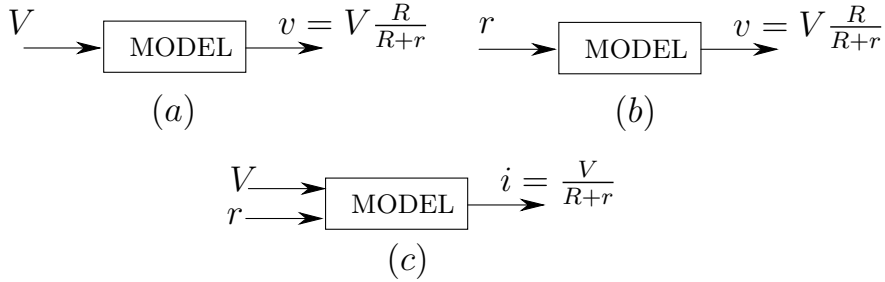


Figure 1.3: Three models for the system of Fig. 1.2.

Depending on what we view as controllable input and observable output, several alternate models of the same underlying system may be constructed.

Example 1.2 (A spring-mass system)

Consider the spring-mass system of Fig. 1.4. Suppose that at time $t = 0$ we displace the mass from its rest position by an amount $u(0) = u_0 > 0$ (the positive direction is as shown in Fig. 1.4) and release it. Let the displacement at any time $t > 0$ be denoted by $y(t)$. We know, from simple mechanics, that the motion of the mass defines a harmonic oscillation described by the second-order differential equation

$$m\ddot{y} = -ky \quad (1.6)$$

with initial conditions $y(0) = u_0$, $\dot{y}(0) = 0$. If we are interested in controlling the initial displacement $u(0)$ and observing the position of the mass as a function of time, we can construct the model shown in Fig. 1.4, where the input is the function $u(t)$ defined so that

$$u(t) = \begin{cases} u_0 & t = 0 \\ 0 & \text{otherwise} \end{cases} \quad (1.7)$$

and the output $y(t)$ is the solution of the differential equation (1.6). Note that the variables k and m are assumed constant.

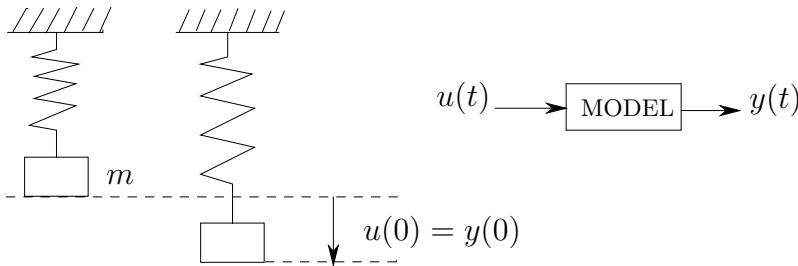


Figure 1.4: Simple mechanical system for Example 1.2 and corresponding model.

Chapter 2

Languages and Automata

2.1 INTRODUCTION

We have seen how discrete-event systems (DES) differ from continuous-variable dynamic systems (CVDS) and why DES are not adequately modeled through differential or difference equations. Our first task, therefore, in studying DES is to develop appropriate models, which both adequately describe the behavior of these systems and provide a framework for analytical techniques to meet the goals of design, control, and performance evaluation.

When considering the state evolution of a DES, our first concern is with the sequence of states visited and the associated events causing these state transitions. To begin with, we will not concern ourselves with the issue of when the system enters a particular state or how long the system remains at that state. We will assume that the behavior of the DES is described in terms of event sequences of the form $e_1e_2\cdots e_n$. A sequence of that form specifies the order in which various events occur over time, but it does not provide the time instants associated with the occurrence of these events. This is the untimed or logical level of abstraction discussed in Sect.1.3.3 in Chap.1, where the behavior of the system is modeled by a *language*. Consequently, our first objective in this chapter is to discuss language models of DES and present operations on languages that will be used extensively in this and the next chapters.

As was mentioned in Sect.1.3.3, the issue of representing languages using appropriate modeling formalisms is key for performing analysis and control of DES. The second objective of this chapter is to introduce and describe the first of the two untimed modeling formalisms for DES considered in this book to represent languages, *automata*. Automata form the most basic class of DES models. As we shall see in this chapter, they are intuitive, easy to use, amenable to composition operations, and amenable to analysis as well (in the finite-state case). On the other hand, they lack structure and for this reason may lead to very large state spaces when modeling complex systems. Nevertheless, any study of discrete event system

and control theory must start with a study of automata. The second modeling formalism considered in this book, Petri nets, will be presented in Chap. 4. As we shall see in that chapter, Petri nets have more structure than automata, although they do not possess, in general, the same analytical power as automata. Other modeling formalisms have been developed for untimed DES, most notably process algebras and logic-based models. These formalisms are beyond the scope of this book; some relevant references are presented at the end of this chapter.

The third objective of this chapter is to present some of the fundamental logical behavior problems we encounter in our study of DES. We would like to have systematic means for fully testing the logical behavior of a system and guaranteeing that it always does what it is supposed to. Using the automaton formalism, we will present solution techniques for three kinds of verification problems, those of safety (i.e., avoidance of illegal behavior), liveness (i.e., avoidance of deadlock and livelock), and diagnosis (i.e., ability to detect occurrences of unobservable events). These are the most common verification problems that arise in the study of software implementations of control systems for complex automated systems. The following chapter will address the problem of *controlling* the behavior of a DES, in the sense of the feedback control loop presented in Sect. 1.2.8, in order to ensure that the logical behavior of the closed-loop system is satisfactory.

Finally, we emphasize that an important objective of this book is to study timed and stochastic models of DES; establishing untimed models constitutes the first stepping stone towards this goal.

2.2 THE CONCEPTS OF LANGUAGES AND AUTOMATA

2.2.1 Language Models of Discrete Event Systems

One of the formal ways to study the logical behavior of DES is based on the theories of languages and automata. The starting point is the fact that any DES has an underlying event set E associated with it. The set E is thought of as the “alphabet” of a language and event sequences are thought of as “words” in that language. In this framework, we can pose questions such as “Can we build a system that speaks a given language?” or “What language does this system speak?”

To motivate our discussion of languages, let us consider a simple example. Suppose there is a machine we usually turn on once or twice a day (like a car, a photocopier, or a desktop computer), and we would like to design a simple system to perform the following basic task: When the machine is turned on, it should first issue a signal to tell us that it is in fact ON, then give us some simple status report (like, in the case of a car, “everything OK”, “check oil”, or “I need gas”), and conclude with another signal to inform us that “status report done”. Each of these signals defines an event, and all of the possible signals the machine can issue define an alphabet (event set). Thus, our system has the makings of a DES driven by these events. This DES is responsible for recognizing events and giving the proper interpretation to any particular sequence received. For instance, the event sequence: “I’m ON”, “everything is OK”, “status report done”, successfully completes our task. On the other hand, the event sequence: “I’m ON”, “status report done”, without some sort of actual status report in between, should be interpreted as an abnormal condition requiring special attention. We can therefore think of the combinations of signals issued by the machine as words belonging to the particular language spoken by this machine. In this particular example, the language of interest should be one with three-event words only,

always beginning with “I’m ON” and ending with “status report done”. When the DES we build sees such a word, it knows the task is done. When it sees any other word, it knows something is wrong. We will return to this type of system in Example 2.10 and see how we can build a simple DES to perform a “status check” task.

Language Notation and Definitions

We begin by viewing the event set E of a DES as an alphabet. We will assume that E is finite. A sequence of events taken out of this alphabet forms a “word” or “string” (short for “string of events”). We shall use the term “string” in this book; note that the term “trace” is also used in the literature. A string consisting of no events is called the empty string and is denoted by ε . (The symbol ε is not to be confused with the generic symbol e for an element of E .) The length of a string is the number of events contained in it, counting multiple occurrences of the same event. If s is a string, we will denote its length by $|s|$. By convention, the length of the empty string ε is taken to be zero.

Definition. (Language)

A *language* defined over an event set E is a set of finite-length strings formed from events in E . ◆

As an example, let $E = \{a, b, g\}$ be the set of events. We may then define the language

$$L_1 = \{\varepsilon, a, abb\} \quad (2.1)$$

consisting of three strings only; or the language

$$L_2 = \{\text{all possible strings of length 3 starting with event } a\} \quad (2.2)$$

which contains nine strings; or the language

$$L_3 = \{\text{all possible strings of finite length which start with event } a\} \quad (2.3)$$

which contains an infinite number of strings.

The key operation involved in building strings, and thus languages, from a set of events E is *concatenation*. The string abb in L_1 above is the concatenation of the string ab with the event (or string of length one) b ; ab is itself the concatenation of a and b . The concatenation uv of two strings u and v is the new string consisting of the events in u immediately followed by the events in v . The empty string ε is the *identity element* of concatenation: $u\varepsilon = \varepsilon u = u$ for any string u .

Let us denote by E^* the set of *all* finite strings of elements of E , including the empty string ε ; the $*$ operation is called the *Kleene-closure*. Observe that the set E^* is countably infinite since it contains strings of arbitrarily long length. For example, if $E = \{a, b, c\}$, then

$$E^* = \{\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, \dots\}$$

A language over an event set E is therefore a *subset* of E^* . In particular, \emptyset , E , and E^* are languages.

We conclude this discussion with some terminology about strings. If $tuv = s$ with $t, u, v \in E^*$, then:

- t is called a *prefix* of s ,

- u is called a *substring* of s , and

- v is called a *suffix* of s .

We will sometimes use the notation s/t (read “ s after t ”) to denote the suffix of s after its prefix t . If t is not a prefix of s , then s/t is not defined.

Observe that both ε and s are prefixes (substrings, suffixes) of s .

Operations on Languages

The usual set operations, such as union, intersection, difference, and complement with respect to E^* , are applicable to languages since languages are sets. In addition, we will also use the following operations:¹

- *Concatenation*: Let $L_a, L_b \subseteq E^*$, then

$$L_a L_b := \{s \in E^* : (s = s_a s_b) \text{ and } (s_a \in L_a) \text{ and } (s_b \in L_b)\}$$

In words, a string is in $L_a L_b$ if it can be written as the concatenation of a string in L_a with a string in L_b .

- *Prefix-closure*: Let $L \subseteq E^*$, then

$$\overline{L} := \{s \in E^* : (\exists t \in E^*) [st \in L]\}$$

In words, the prefix closure of L is the language denoted by \overline{L} and consisting of all the prefixes of all the strings in L . In general, $L \subseteq \overline{L}$.

L is said to be *prefix-closed* if $L = \overline{L}$. Thus language L is prefix-closed if any prefix of any string in L is also an element of L .

- *Kleene-closure*: Let $L \subseteq E^*$, then

$$L^* := \{\varepsilon\} \cup L \cup LL \cup LLL \cup \dots$$

This is the same operation that we defined above for the set E , except that now it is applied to set L whose elements may be strings of length greater than one. An element of L^* is formed by the concatenation of a finite (but possibly arbitrarily large) number of elements of L ; this includes the concatenation of “zero” elements, that is, the empty string ε . Note that the $*$ operation is idempotent: $(L^*)^* = L^*$.

- *Post-language*: Let $L \subseteq E^*$ and $s \in L$. Then the post-language of L after s , denoted by L/s , is the language

$$L/s := \{t \in E^* : st \in L\}$$

By definition, $L/s = \emptyset$ if $s \notin \overline{L}$.

Observe that in expressions involving several operations on languages, prefix-closure and Kleene-closure should be applied first, and concatenation always precedes operations such as union, intersection, and set difference. (This was implicitly assumed in the above definition of L^* .)

¹ “ \equiv ” denotes “equal to by definition.”

Example 2.1 (Operations on languages)

Let $E = \{a, b, g\}$, and consider the two languages $L_1 = \{\varepsilon, a, abb\}$ and $L_4 = \{g\}$. Neither L_1 nor L_4 are prefix-closed, since $ab \notin L_1$ and $\varepsilon \notin L_4$. Then:

$$\begin{aligned} L_1 L_4 &= \{g, ag, abbg\} \\ \overline{L_1} &= \{\varepsilon, a, ab, abb\} \\ \overline{L_4} &= \{\varepsilon, g\} \\ L_1 \overline{L_4} &= \{\varepsilon, a, abb, g, ag, abbg\} \\ L_4^* &= \{\varepsilon, g, gg, ggg, \dots\} \\ L_1^* &= \{\varepsilon, a, abb, aa, aabb, abba, abbabb, \dots\} \end{aligned}$$

We make the following observations for technical accuracy:

- (i) $\varepsilon \notin \emptyset$;
- (ii) $\{\varepsilon\}$ is a nonempty language containing only the empty string;
- (iii) If $L = \emptyset$ then $\overline{L} = \emptyset$, and if $L \neq \emptyset$ then necessarily $\varepsilon \in \overline{L}$;
- (iv) $\emptyset^* = \{\varepsilon\}$ and $\{\varepsilon\}^* = \{\varepsilon\}$;
- (v) $\emptyset L = L \emptyset = \emptyset$.

Projections of Strings and Languages

Another type of operation frequently performed on strings and languages is the so-called *natural projection*, or simply *projection*, from a set of events, E_l , to a *smaller* set of events, E_s , where $E_s \subset E_l$. Natural projections are denoted by the letter P ; a subscript is typically added to specify either E_s or both E_l and E_s for the sake of clarity when dealing with multiple sets. In the present discussion, we assume that the two sets E_l and E_s are fixed and we use the letter P without subscript.

We start by defining the projection P for strings:

$$P : E_l^* \rightarrow E_s^*$$

where

$$\begin{aligned} P(\varepsilon) &:= \varepsilon \\ P(e) &:= \begin{cases} e & \text{if } e \in E_s \\ \varepsilon & \text{if } e \in E_l \setminus E_s \end{cases} \\ P(se) &:= P(s)P(e) \text{ for } s \in E_l^*, e \in E_l \end{aligned}$$

As can be seen from the definition, the projection operation takes a string formed from the larger event set (E_l) and *erases* events in it that do not belong to the smaller event set (E_s).

We will also be working with the corresponding inverse map

$$P^{-1} : E_s^* \rightarrow 2^{E_l^*}$$

defined as follows

$$P^{-1}(t) := \{s \in E_l^* : P(s) = t\}$$

Chapter 5

Timed and Hybrid Models

5.1 INTRODUCTION

In this chapter we concentrate on timed models of DES. We also explore what happens when a system combines time-driven dynamics with event-driven dynamics giving rise to what are referred to as *hybrid systems*, which were introduced at the end of Chap. 1. The simplest instance of time-driven dynamics is the case of adjoining one or more clocks to the untimed DES model, resulting in a *timed model*. In the case of timed DES models, the sample paths are no longer specified as event sequences $\{e_1, e_2, \dots\}$ or state sequences $\{x_0, x_1, \dots\}$, but they must include some form of timing information. For example, let t_k , $k = 1, 2, \dots$, denote the time instant when the k th event and state transition occurs (with t_0 given); then a timed sample path of a DES may be described by the sequence $\{(x_0, t_0), (x_1, t_1), \dots\}$. Similarly, a timed sequence of events would look like $\{(e_1, t_1), (e_2, t_2), \dots\}$. Creating a framework for timed DES models will enable us to address questions such as “how many events of a particular type can occur in a given time interval?”, “is the time interval between occurrences of two different events always greater than a given lower bound?” or “how long does the system spend in a given state?” These issues are of critical importance in analyzing the behavior of many classes of DES since they provide us with particularly useful measures of system performance. The next step is to consider more complicated time-driven dynamics than clocks. Namely, at each discrete state of the system, we can associate a set of differential equations that describe the evolution of continuous variables of interest. This brings us to the realm of hybrid systems.

The first objective of this chapter is to describe timing mechanisms for DES. For this purpose, we will introduce the notion of a *clock structure*, which will capture the timing constraints associated with the consecutive occurrences of each event in the model and will serve as input to the untimed model. Using the mechanism of a clock structure, we will find that we can directly extend the untimed automata and Petri nets that were introduced

in previous chapters to create modeling frameworks for DES that include event timing. Establishing the framework of *timed automata* and *timed Petri nets* with clock structures is our second objective (Sects. 5.2 and 5.3). Most of the analytical techniques in subsequent chapters will be based on timed automata with clock structures (or simply timed automata). Another approach we will briefly cover uses a special kind of algebra, referred to as a *dioid algebra*, and attempts to parallel some of the analytical techniques used for linear CVDS for some classes of timed DES (Sect. 5.4). The third objective of this chapter is to present alternative timed models of DES (Sects. 5.5 and 5.6). In particular, *timed automata with guards* will be defined (Sect. 5.6). This modeling framework is known under the name of Alur and Dill timed automata, from the two researchers who proposed this model in the early 1990s. Timed automata with guards employ a generalized form of clock structure where a set of clocks with time-driven dynamics are adjoined to the automaton and the transitions of the automaton include pre-conditions on the clock values, known as *guards*. Timed automata with guards form the basis of *hybrid automata*, which include more complicated time-driven dynamics than clocks. The last objective of this chapter is to introduce the notion of a hybrid automaton along with some basic associated concepts on hybrid systems (Sect. 5.7).

As will become clear upon reading this chapter, all input information required for the timed models we develop is assumed to be available, either in the form of a clock structure or in the form of timing intervals. This means that we have a means of describing in a non-stochastic manner *when* events are allowed to occur in the future. Of course, it is unrealistic to expect this to be always the case. A computer system user does not announce the jobs he will submit and their times of submission for the next week. Nor does a common person plan days in advance when he is going to place various telephone calls. In short, we will ultimately have to resort to probabilistic models in order to carry out some analysis of practical value; this is going to be the goal of the next chapter. We limit ourselves in this chapter to an input description of timing considerations which is specified in a non-stochastic manner, so as to gain an understanding of the basic event timing dynamics of a DES, independent of the probabilistic characterization of its input.

5.2 TIMED AUTOMATA

We begin with an automaton model $G = (X, E, f, \Gamma, x_0)$, where

X	is a countable <i>state space</i>
E	is a countable <i>event set</i>
$f : X \times E \rightarrow X$	is a <i>state transition function</i> and is generally a <i>partial</i> function on its domain
$\Gamma : X \rightarrow 2^E$	is the <i>active event function</i> (or feasible event function); $\Gamma(x)$ is the set of all events e for which $f(x, e)$ is defined and it is called the <i>active event set</i> (or feasible event set)
x_0	is the <i>initial state</i>

This is the same as the automaton defined in Chap. 2 with only a few minor changes. First, we allow for generally countable (as opposed to finite) sets X and E . We also leave out of the definition any consideration for marked states, since we will not be considering blocking issues in this chapter.

5.2.1 The Clock Structure

In order to introduce the key ideas of the timing mechanism we need, we start out by discussing the simplest possible DES we can think of. We then gradually proceed with more complicated cases.

A DES with a single event. Let $E = \{\alpha\}$, and $\Gamma(x) = \{\alpha\}$ for all $x \in X$. A sample path of this simple system on the time line is shown in Fig. 5.1. The event sequence associated with this sample path is denoted by $\{e_1, e_2, \dots\}$, where $e_k = \alpha$ for all $k = 1, 2, \dots$. The time instant associated with the k th occurrence of the event is denoted by t_k , $k = 1, 2, \dots$. The length of the time interval defined by two successive occurrences of the event is called a *lifetime*. Thus, we define

$$v_k = t_k - t_{k-1} \quad k = 1, 2, \dots \quad (5.1)$$

to be the k th lifetime of the event. Clearly, this is a nonnegative real number, i.e., $v_k \in \mathbb{R}^+$.

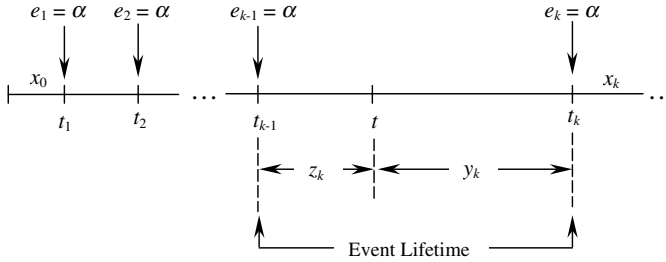


Figure 5.1: Sample path of a DES with $E = \{\alpha\}$.

The evolution of this system over time can be described as follows. At time t_{k-1} , the k th event is said to be *activated* or *enabled*, and is given a lifetime v_k . A clock associated with the event is immediately set at the value specified by v_k , and then it starts ticking down to 0. During this time interval, the k th event is said to be *active*. The clock reaches zero when the lifetime expires at time $t_k = t_{k-1} + v_k$. At this point, the event has to occur. This will cause a state transition. The process then repeats with the $(k+1)$ th event becoming active.

Observe the difference between the event being “active” and the event “occurring.” The event α is active as long as it is feasible in the current state, that is, $\alpha \in \Gamma(x)$. The event actually occurs when its clock runs down to 0, and a state transition takes place. This is similar to the distinction between “enabling” a transition and “firing” it in Petri nets, as discussed in the last chapter. In Fig. 5.1, the event α is in fact continuously active, but it only occurs at the time instants t_1, t_2, \dots . Every time it occurs, it is immediately activated anew, since it is always feasible in this example.

To introduce some further notation, let t be any time instant, not necessarily associated with an event occurrence. Suppose $t_{k-1} \leq t \leq t_k$. Then t divides the interval $[t_{k-1}, t_k]$ into two parts (see Fig. 5.1) such that

$$y_k = t_k - t \quad (5.2)$$

is called the *clock* or *residual lifetime* of the k th event, and

$$z_k = t - t_{k-1} \quad (5.3)$$

is called the *age* of the k th event. It is immediately obvious that

$$v_k = z_k + y_k \quad (5.4)$$

It should be clear that a sample path of this DES is completely specified by the lifetime sequence $\{v_1, v_2, \dots\}$. This is also referred to as the *clock sequence* of event α .

A DES with two permanently active events. Things become more interesting if we consider a DES with $E = \{\alpha, \beta\}$. For simplicity, we first assume that $\Gamma(x) = \{\alpha, \beta\}$ for all $x \in X$, so that both events are always active. Suppose that a clock sequence for each event is specified, that is, there are two known sequences of lifetimes:

$$\mathbf{v}_\alpha = \{v_{\alpha,1}, v_{\alpha,2}, \dots\}, \quad \mathbf{v}_\beta = \{v_{\beta,1}, v_{\beta,2}, \dots\}$$

Starting with a given time t_0 , the first question we address is: *Which event occurs next?* It is reasonable to answer this question by comparing $v_{\alpha,1}$ to $v_{\beta,1}$ and selecting the event with the shortest lifetime. Thus, if

$$v_{\alpha,1} < v_{\beta,1}$$

α is the first event to occur at time $t_1 = t_0 + v_{\alpha,1}$, as illustrated in Fig. 5.2.

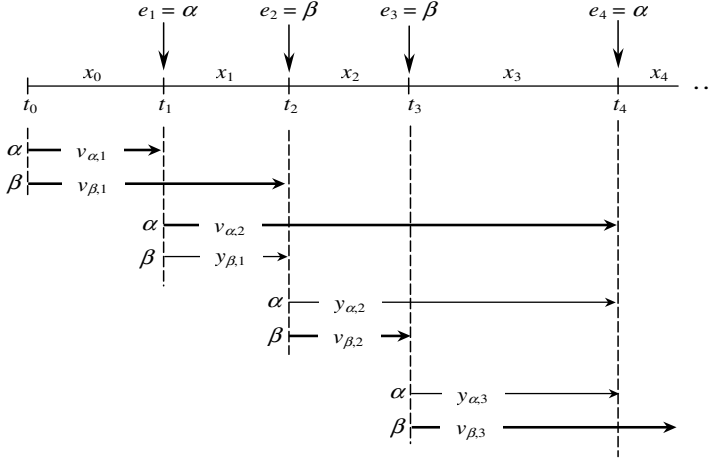


Figure 5.2: Sample path of a DES with $E = \{\alpha, \beta\}$ and both events permanently active. *The next event to occur is always the one with the smallest clock value. When an event occurs, it is immediately activated again, and its clock is reset to the next available lifetime value in its assigned clock sequence.*

We are now at time t_1 , and we may pose the same question once again: *Which event occurs next?* Since β is still active, it has a clock value given by

$$y_{\beta,1} = v_{\beta,1} - v_{\alpha,1}$$

On the other hand, α is also active. Since α just took place, its next occurrence is defined by the new lifetime $v_{\alpha,2}$, taken out of the given clock sequence. We now compare $v_{\alpha,2}$ to

$y_{\beta,1}$ and select the smaller of the two. Thus, if

$$y_{\beta,1} < v_{\alpha,2}$$

β is the second event to occur at time $t_2 = t_1 + y_{\beta,1}$, as illustrated in Fig. 5.2.

The mechanism for selecting the “next event” at any time is therefore based on *comparing clock values and selecting the smallest one*. If an event has just occurred, then its clock is actually reset to a new lifetime value supplied by its clock sequence.

A DES with two not permanently active events. Our modeling framework would not be interesting if no events could ever be made inactive. Suppose that in the previous DES, $\Gamma(x) = \{\alpha, \beta\}$ for some $x \in X$, and $\Gamma(x) = \{\beta\}$ for all remaining $x \in X$. For a given initial state x_0 , let $\Gamma(x_0) = \{\alpha, \beta\}$. Thus, as before, the first event to occur is α (see Fig. 5.3). Next, let us assume that the new state, x_1 , is such that $\Gamma(x_1) = \{\beta\}$. In this case, the only remaining feasible event is β , and no comparison with any other clock is required. Next, when β occurs at time t_2 , it causes a state transition into x_2 . Suppose both α and β are feasible once again. In this case, both events are activated. We compare their clock values (the new lifetimes assigned to them at this point), and find that $v_{\beta,2} < v_{\alpha,2}$, so event β occurs next (see Fig. 5.3). Now suppose that the new state, x_3 , is such that $\Gamma(x_3) = \{\beta\}$. Since α is no longer feasible, it is *deactivated* or *disabled*. Its clock becomes irrelevant in the determination of the next event, and is therefore ignored. Finally, at time t_4 , β occurs and causes a transition to a new state x_4 . Assuming $\Gamma(x_4) = \{\alpha, \beta\}$, both events are once again activated. Note that a completely new lifetime, $v_{\alpha,3}$, is selected for event α , since $v_{\alpha,2}$ was discarded when α was deactivated at t_3 .

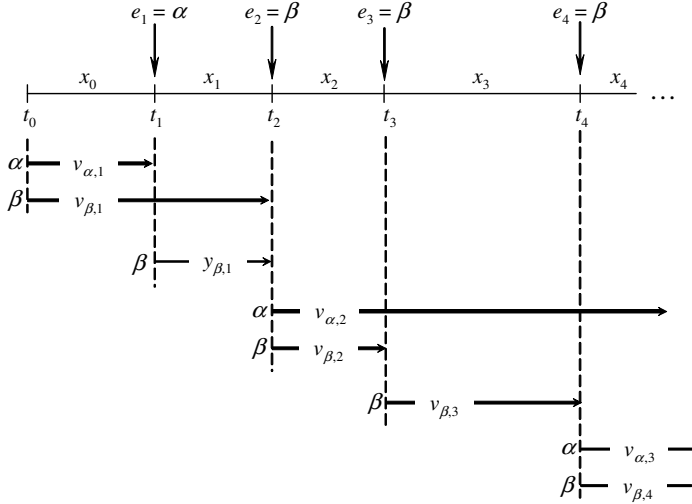


Figure 5.3: Sample path of a DES with $E = \{\alpha, \beta\}$ and α not feasible in some states. In this example, α is not feasible in states x_1 and x_3 . At t_1 , α is not activated again after it occurs. At t_3 , α is explicitly deactivated when β occurs. The clock value of α at this point is discarded. When it is activated again at t_4 , a new lifetime is assigned to the clock of α .

Therefore, in general, the mechanism for selecting the “next event” at any time is based on three simple rules:



<http://www.springer.com/978-0-387-33332-8>

Introduction to Discrete Event Systems

Cassandras, C.G.; Lafortune, S.

2008, XXIV, 772 p., Hardcover

ISBN: 978-0-387-33332-8