1

Systems

The word system is used in this book as a synonym for *mathematical model of a dynamical phenomenon*. Since different problems may require different models of the same phenomenon, we need a versatile notion of system that can be equipped with relationships explaining how different systems can be related. The purpose of this chapter is to provide one such notion and to illustrate its use in different contexts.

Notation

For a set Z, $1_Z : Z \to Z$ denotes the identity map on Z defined by $1_Z(z) = z$ for every $z \in Z$. Given a map $f : Z \to W$ and a set $K \subseteq Z$, $f(K)$ denotes the subset of W defined by $f(K) = \{w \in W \mid w = f(k) \text{ for some } k \in K\}$ while $f|_K : K \to W$ describes the restriction of f to K defined by $f|_K(k) = f(k)$ for every $k \in K$. The projection map taking $(x_a, x_b, u_a, u_b) \in X_a \times X_b \times U_a \times U_b$ to $(x_a, x_b) \in X_a \times X_b$ is denoted by π_X.

1.1 System definition

Among the many different mathematical models used to describe dynamical phenomena we are especially interested in models with states belonging to finite sets, infinite sets, and combinations thereof. By a finite-state system we mean a system described by finitely many states. The finite-state machines used to model digital circuits are one such example. We also consider infinite-state systems described by difference or differential equations with solutions evolving in infinite sets such as $\mathbb{R}^n$. Hybrid systems, combining aspects of finite-state and infinite-state systems, consist of another class of systems that can be described by the notion of system adopted in this book.

P. Tabuada, *Verification and Control of Hybrid Systems: A Symbolic Approach*,
DOI: 10.1007/978-1-4419-0224-5_1, © Springer Science + Business Media, LLC 2009

Definition 1.1 (System). *A* system S *is a sextuple* $(X, X_0, U, \longrightarrow, Y, H)$ *consisting of:*

- *a set of* states X*;*
- *a set of* initial states $X_0 \subseteq X$*;*
- *a set of* inputs U*;*
- *a* transition relation $\longrightarrow \subseteq X \times U \times X$*;*
- *a set of* outputs Y*;*
- *an* output map $H : X \to Y$*.*

States in X are regarded as internal to the system whereas outputs are externally visible. The set of initial states may be a proper subset $X_0 \subset X$, a fixed initial state $x_0 \in X$, or the whole set of states $X_0 = X$. A system is called *finite-state* if X is a finite set. A system that is not finite-state is called *infinite-state*. Systems described by differential equations are examples of infinite-state systems. The adjectives finite and infinite always qualify the state set of a system whereas the adjectives *discrete* and *continuous* are used to qualify time. Even though a state cannot[1] be qualified as finite or infinite, we shall abuse language and call a state finite or infinite when the corresponding state set is finite or infinite, respectively. This abuse of language will be extremely useful throughout the book. The relationship between finite-state and infinite-state systems is an important topic that is considered in great detail in this book.

The evolution of a system is captured by the transition relation. A transition $(x, u, x') \in \longrightarrow$ is, throughout the book, denoted by $x \xrightarrow{u} x'$. For such a transition, state x' is called a *u-successor*, or simply *successor*, of state x. Similarly, x is called a *u-predecessor*, or *predecessor*, of state x'. Note that, since $\longrightarrow \subseteq X \times U \times X$ is a relation, for any state and any input $u \in U$ there may be: no u-successors, one u-successor, or many u-successors. For conciseness, we denote the set of u-successors of a state x by $\mathrm{Post}_u(x)$. Since $\mathrm{Post}_u(x)$ may be empty, we denote by $U(x)$ the set of inputs $u \in U$ for which $\mathrm{Post}_u(x)$ is nonempty. As discussed in later chapters, the semantics of the elements in U depends on the problem being solved. Inputs in U can represent choices to be made by a controller, choices to be made by the environment, or they can simply describe the passage of time.

Example 1.2. Finite-state systems naturally arise as models of a variety of man-made phenomena. In addition to being completely defined by the data described in Definition 1.1, they also admit a graphical representation that is very useful. States are represented by circles and transitions are represented by arrows between states. Initial states are distinguished by being the target of a sourceless arrow. Each circle is labeled by the state (top half) and the

[1] A state is simply an element of the set of states X. Therefore, unless additional structure is imposed on X, the expressions "infinite state" and "finite state" have no defined meaning.

corresponding output (bottom half), and each arrow is labeled by the input. The graphical representation of the finite-state system defined by the data:

$$X = \{x_0, x_1, x_2, x_3\}, \quad X_0 = \{x_0, x_2\}, \quad U = \{u_0, u_1\}, \tag{1.1}$$

$$\longrightarrow = \{(x_0, u_0, x_1), (x_0, u_1, x_2), (x_1, u_0, x_1), (x_1, u_0, x_3), (x_2, u_1, x_3), (x_3, u_1, x_1)\}, \tag{1.2}$$

$$Y = \{y_0, y_1, y_2\}, \tag{1.3}$$

$$H(x_0) = y_0, \quad H(x_1) = y_0, \quad H(x_2) = y_1, \quad H(x_3) = y_2, \tag{1.4}$$

is displayed in Figure 1.1. ◁

A system is called *blocking* if there is a state $x \in X$ from which no further transitions are possible, *i.e.*, x has no u-successors for any $u \in U$. This can also be expressed as $U(x) = \varnothing$. A system is called *nonblocking* if the set of successors of every $x \in X$ is nonempty. An equivalent characterization is $U(x) \neq \varnothing$ for every $x \in X$.

A system is called *deterministic* if for any state $x \in X$ and any input $u \in U$, $x \xrightarrow{u} x'$ and $x \xrightarrow{u} x''$ imply $x' = x''$. Therefore, a system is deterministic if given any state $x \in X$ and any input $u \in U$, there exists at most one u-successor (there may be none). A system is *output deterministic* if: $H|_{X_0}$ is injective; and for any state $x \in X$ and any inputs $u, u' \in U$, $x \xrightarrow{u} x'$ and $x \xrightarrow{u'} x''$ with $H(x') = H(x'')$ imply $x' = x''$. For output deterministic systems, different successors of a state always have different outputs.

A system is called *nondeterministic* if it is not deterministic. Hence for a nondeterministic system it is possible for a state to have two (or possibly more) distinct u-successors.

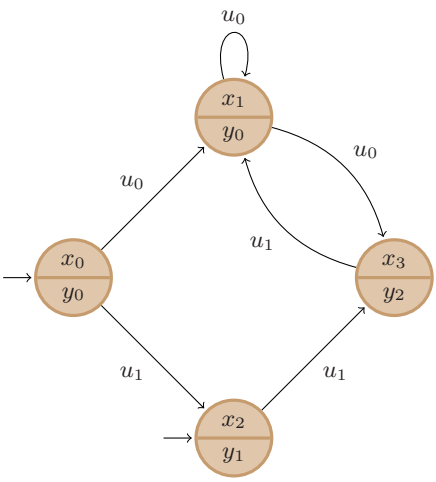

Fig. 1.1. Graphical representation of the finite-state system defined by (1.1) through (1.4).

One can easily see that the system represented in Figure 1.1 is nonblocking since every state has an outgoing transition. It is also nondeterministic as there are two u_0-successors of the state x_1, namely x_1 and x_3. Albeit not being deterministic this system is output deterministic.

To simplify notation we also denote a system $S = (X, X_0, U, \longrightarrow, Y, H)$ by the quintuple $S = (X, U, \longrightarrow, Y, H)$ when $X_0 = X$, by the quadruple $S = (X, X_0, U, \longrightarrow)$ when $Y = X$ and $H = 1_X$, or by the triple $S = (X, U, \longrightarrow)$ when $X_0 = X = Y$ and $H = 1_X$.

1.2 System behavior

Given any state $x \in X$, a *finite internal behavior* generated from x is a finite sequence of transitions:

$$x_0 \xrightarrow{u_0} x_1 \xrightarrow{u_1} x_2 \xrightarrow{u_2} \dots \xrightarrow{u_{n-2}} x_{n-1} \xrightarrow{u_{n-1}} x_n$$

such that $x_0 = x$ and $x_i \xrightarrow{u_i} x_{i+1}$ for all $0 \leq i < n$. A state $x \in X$ can also be seen as a behavior comprising zero transitions in which case $n = 0$. An internal behavior generated from x is *initialized* if $x \in X_0$.

In some cases, a finite internal behavior can be extended to an infinite internal behavior. An *infinite internal behavior* generated from x is an infinite sequence:

$$x_0 \xrightarrow{u_0} x_1 \xrightarrow{u_1} x_2 \xrightarrow{u_2} x_3 \xrightarrow{u_3} \dots$$

that satisfies $x_0 = x$ and $x_i \xrightarrow{u_i} x_{i+1}$ for all $i \in \mathbb{N}_0$. An infinite internal behavior generated from x is called *initialized* if $x \in X_0$. In nonblocking systems, every finite internal behavior can be extended to an infinite internal behavior.

Through the output map, every internal behavior:

$$x_0 \xrightarrow{u_0} x_1 \xrightarrow{u_1} x_2 \xrightarrow{u_2} \dots \xrightarrow{u_{n-2}} x_{n-1} \xrightarrow{u_{n-1}} x_n$$

defines an external behavior:

$$y_0 \longrightarrow y_1 \longrightarrow y_2 \longrightarrow \dots \longrightarrow y_{n-1} \longrightarrow y_n \tag{1.5}$$

with $H(x_i) = y_i \in Y$ for all $0 \leq i \leq n$. We also use the more succinct notation $\mathsf{y} = y_0 y_1 y_2 \dots y_n$ to represent the external behavior (1.5). The set of external behaviors that are defined by internal behaviors generated from state x is denoted by $\mathcal{B}_x(S)$ and is called the *external behavior* from state x.

Definition 1.3 (Finite External Behavior). *The* finite external behavior *generated by a system S, denoted by $\mathcal{B}(S)$, is defined by:*

$$\mathcal{B}(S) = \bigcup_{x \in X_0} \mathcal{B}_x(S).$$

For output deterministic systems any finite external behavior $\mathtt{y}$ determines uniquely the corresponding internal behavior. This can easily be shown by induction. Given an external behavior $\mathtt{y} = y_0 y_1 y_2 \dots y_n$ we can recover the corresponding initial state x_0 since $H|_{X_0}$ is injective. Then, we consider all the successors x_1 of x_0 satisfying $H(x_1) = y_1$. If there is more than one successor, say x_1 and x_1', we have $x_0 \xrightarrow{u_1} x_1$ and $x_0 \xrightarrow{u_1'} x_1'$ with $H(x_1) = H(x_1')$. It follows by output determinism that $x_1 = x_1'$ and x_1 is uniquely determined. Applying the same argument to the successors of x_1 we can uniquely recover x_2 and so on.

An infinite internal behavior from x:

$$x_0 \xrightarrow{u_0} x_1 \xrightarrow{u_1} x_2 \xrightarrow{u_2} x_3 \xrightarrow{u_3} \dots$$

defines an infinite external behavior:

$$y_0 \longrightarrow y_1 \longrightarrow y_2 \longrightarrow y_3 \longrightarrow \dots \tag{1.6}$$

corresponding to the infinite sequence of outputs with $H(x_i) = y_i$ for all $i \in \mathbb{N}_0$. The infinite external behavior (1.6) can also be succinctly denoted by $\mathtt{y} = y_0 y_1 y_2 y_3 \dots$. The set of all infinite external behaviors that are generated from x is denoted by $\mathcal{B}_x^\omega(S)$ and called the *infinite external behavior* from state x.

Definition 1.4 (Infinite External Behavior). *The* infinite external behavior *generated by a system S, denoted by $\mathcal{B}^\omega(S)$, is defined by:*

$$\mathcal{B}^\omega(S) = \bigcup_{x \in X_0} \mathcal{B}_x^\omega(S).$$

Infinite behaviors describe the nonterminating interaction of a system with other systems and the environment. They are thus adequate to model the operation of reactive systems, such as embedded controllers, that must operate without interruption for arbitrarily long periods of time. For this reason, we focus mostly on infinite behaviors and drop the adjective infinite whenever clear from the context.

If a system S is non-blocking, then $\mathcal{B}^\omega(S)$ is nonempty. However, $\mathcal{B}^\omega(S)$ may be nonempty even if S is a blocking system. Figure 1.2 displays one such example where the infinite external behavior $\mathtt{aaaaa}\dots$ belongs to $\mathcal{B}^\omega(S)$ although S is blocking since the state x_1 has no successors.

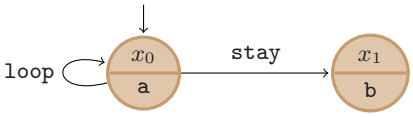

Fig. 1.2. Graphical representation of a blocking finite-state system with nonempty infinite external behavior.

In Chapter 4 we discuss in further detail the relation between $\mathcal{B}(S)$, $\mathcal{B}^\omega(S)$, and the blocking/nonblocking properties of S. For now, we veer to the relation between inputs and outputs. In many situations the inputs provide valuable information that is not directly captured by the internal or external behavior of a system. This can be easily remedied by suitably extending the state set. Starting from a system $S = (X, X_0, U, \longrightarrow, Y, H)$ we can construct a new system $S_o = (X_o, X_{o0}, U_o, \underset{o}{\longrightarrow}, Y_o, H_o)$ with:

- $X_o = X \times U$;
- $X_{o0} = X_0 \times \{*\}$ for some element $* \notin U$;
- $U_o = U \cup \{*\}$;
- $(x, u) \underset{o}{\overset{u'}{\longrightarrow}} (x', u')$ in S_o if $x \overset{u'}{\longrightarrow} x'$ in S;
- $Y_o = Y \times U$;
- $H_o(x, u) = (H(x), u)$.

An infinite external behavior of S_o is of the form:

$$(y_0, *) \longrightarrow (y_1, u_0) \longrightarrow (y_2, u_1) \longrightarrow (y_3, u_2) \longrightarrow \dots$$

thus containing not only the infinite external behavior of S:

$$y_0 \longrightarrow y_1 \longrightarrow y_2 \longrightarrow y_3 \longrightarrow \dots \tag{1.7}$$

but also the sequence of inputs $u_0 u_1 u_2 \dots$ used to generated (1.7) in S. Hence, the output set Y and output map H can be designed to make externally visible the aspects (inputs and states) of a system that are considered relevant for the verification or control problem being solved.

1.3 Examples

The examples that follow illustrate the versatility of the notion of system adopted in this book. The results presented in Part II, III, and IV apply not only to the examples in this section but also to many other examples that can be suitably described by the adopted notion of system.

1.3.1 Finite-state systems

Communication protocol

As a first example of finite-state systems we model a very simple communication protocol. Consider a sender and a receiver exchanging messages over an unreliable communication channel. The sender obtains the data to be transmitted from a buffer and sends it through the channel. Since the channel is unreliable, the sender waits for a confirmation message from the receiver. If the confirmation message acknowledges a correct reception, new data is fetched

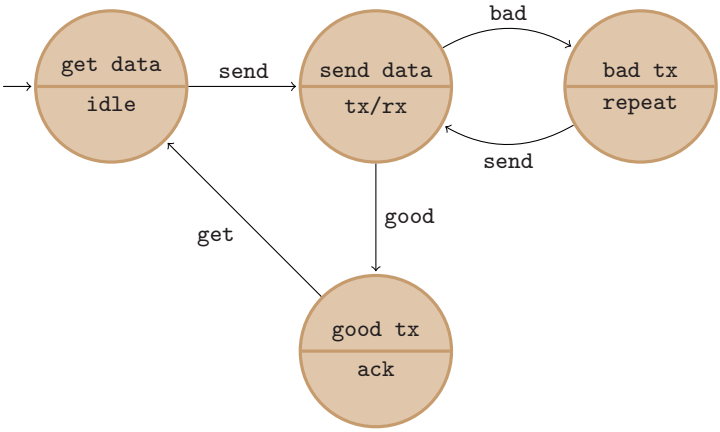

Fig. 1.3. Graphical representation of the finite-state system modeling the sender.

and transmitted. When the confirmation message acknowledges an incorrect reception, the previous message is resent. The behavior of the sender can be described by the finite-state system in Figure 1.3.

The communication channel can either deliver the sent message without errors or deliver a corrupted version of the sent message. The communication channel is not independently modeled but is integrated in the model of the sender and the model of the receiver.

The receiver sends a confirmation message acknowledging the reception of an error free message or a repeat request when the received message is corrupted. The receiver is described by the finite-state system in Figure 1.4.

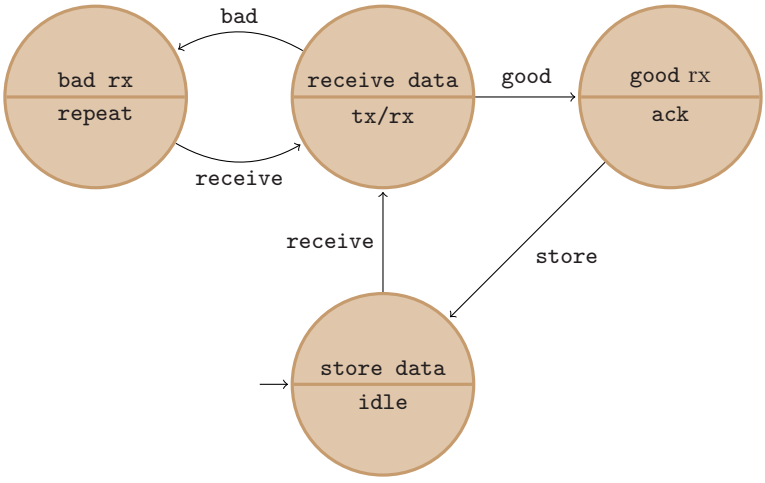

Fig. 1.4. Graphical representation of the finite-state system modeling the receiver.

Although we have modeled the sender and transmitter individually, any analysis regarding the correctness of this protocol is based on the concurrent execution of these two systems. In Section 1.4 we will see how we can compose individual systems into a single system describing its concurrent evolution. For now, we note that both the sender and receiver systems are nonblocking and deterministic. Possible external behaviors of the system modeling the receiver are:

idle ⟶ tx/rx ⟶ ack ⟶ idle ⟶ tx/rx ⟶ repeat
⟶ tx/rx ⟶ repeat ⟶ tx/rx ⟶ ack ⟶ idle

idle ⟶ tx/rx ⟶ repeat ⟶ tx/rx ⟶ repeat ⟶ tx/rx
⟶ repeat ⟶ tx/rx ⟶ repeat ⟶ tx/rx ⟶ repeat
⟶ tx/rx ⟶ repeat ⟶ tx/rx ⟶ repeat ⟶ ...

idle ⟶ tx/rx ⟶ repeat ⟶ tx/rx ⟶ ack ⟶ idle
⟶ tx/rx ⟶ repeat ⟶ tx/rx ⟶ ack ⟶ idle ⟶ tx/rx

Software

Finite-state systems can also be used to model software. Intuitively, we can regard the memory contents of a computational device as its state and the software as a description of how memory contents change over time. Since we can only store finitely many bits of information in the memory of a computing system, the set of states is necessarily finite.

Constructing finite-state models of software is a challenging task that we shall not address in this book. Instead, we provide a very simple example.

Example 1.5. The computation of averages is a problem that occurs frequently in applications. Suppose that we want to compute the average of a stream of numbers but we do not know a priori the length of the stream. One possible way to compute the average is to update the average upon the reception of a new number in the stream. This idea is implemented by Algorithm 1.1 where y contains the latest received number and x contains the average of the numbers that have been received so far. Assume now that we are interested in knowing if x is smaller, equal, or greater than 1 when y is restricted to assume values

x:=0;
n:=0;
while *true* **do**
  y:=read(input);
  $x := x\frac{n}{n+1} + y\frac{1}{n+1}$;
  $n := n + 1$;
end

Algorithm 1.1: Average of a stream of numbers.

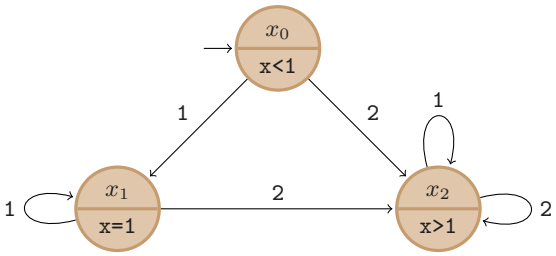

Fig. 1.5. Finite-state system describing if $x < 1$, $x = 1$, or $x > 1$ when $y \in \{1, 2\}$ and x evolves according to Algorithm 1.1.

in the set $\{1, 2\}$. One possible finite-state model capturing the dynamics of x is represented in Figure 1.5. Every execution of the while loop is captured by a transition of the system. $\lhd$

1.3.2 Infinite-state systems

Dynamical systems

Paul Samuelson introduced in 1939 the following model for the evolution of the national income:

$$\begin{aligned} y(n) &= c(n) + i(n) + g(n) \\ c(n+1) &= \alpha y(n) \\ i(n+1) &= \beta(c(n+1) - c(n)) \end{aligned}$$

where α and γ are parameters and y denotes the national income that is formed by three different kinds of expenditures: consumption, denoted by c; investment, denoted by i; and government expenditure, denoted by g. Eliminating y we obtain:

$$\begin{aligned} c(n+1) &= \alpha\big(c(n) + i(n) + g(n)\big) \\ i(n+1) &= \beta\alpha\big(c(n) + i(n) + g(n)\big) - \beta c(n). \end{aligned}$$

If we assume that g is a fixed value, the preceding model defines the system S with:

- $X = (\mathbb{R}_0^+)^2$;
- $X_0 = X$;
- $U = \{*\}$;
- $(c, i) \xrightarrow{*} (c', i')$ iff $c' = \alpha(c + i + g)$ and $i' = \beta\alpha(c + i + g) - \beta c$;
- $Y = \mathbb{R}_0^+$;
- $H(c, i) = c + i + g$.

For discrete-time dynamical systems, the input $*$ is merely used to advance the current state to the uniquely defined next state. Hence, system S is nonblocking and deterministic.

Dynamical systems evolving in continuous-time can also be modeled as systems. Consider Euler's equations for the rotational dynamics of a rigid body around its center of mass:

$$\frac{d}{dt}\xi_1 = \frac{I_2 - I_3}{I_1}\xi_2\xi_3 \tag{1.8}$$

$$\frac{d}{dt}\xi_2 = \frac{I_3 - I_1}{I_2}\xi_3\xi_1 \tag{1.9}$$

$$\frac{d}{dt}\xi_3 = \frac{I_1 - I_2}{I_3}\xi_1\xi_2 \tag{1.10}$$

where $\xi = (\xi_1, \xi_2, \xi_3)$ is the body angular velocity and $I_1, I_2, I_3 \in \mathbb{R}$ are the principal moments of inertia. A solution to these equations with initial condition $x_0 = (x_{10}, x_{20}, x_{30})$ is a continuously differentiable curve $\xi :]a, b[\to \mathbb{R}^3$ with $a < 0 < b$, satisfying $\xi(0) = x_0$ and (1.8) through (1.10) for all $t \in]a, b[$. We can thus model the preceding continuous-time dynamical system as a system S with:

- $X = \mathbb{R}^3$;
- $X_0 = X$;
- $U = \mathbb{R}_0^+$;
- $x \xrightarrow{\tau} x'$ if there exists a solution ξ to equations (1.8) through (1.10) satisfying $\xi(0) = x$ and $\xi(\tau) = x'$;
- $Y = X$;
- $H = 1_X$.

The preceding construction encodes in the transition relation of S all the information contained in the solution ξ since there exists a transition $x \xrightarrow{\tau} x'$ in S iff $\xi(0) = x$ and $\xi(\tau) = x'$. Although this is natural from a mathematical point of view, some readers may feel unsettled with this modeling choice: by labeling the transitions with time we are formally treating time as the input of S. If one thinks of dynamics as change in space corresponding to change in time, then one can easily reconcile oneself with the idea of treating time as an input. Although one can conceive the existence of a uniform notion of time across all systems, different interactions with a system may require different time inputs. As an example, consider two different digital platforms measuring the output of system S. If platform A has a clock cycle of 2 time units and platform B has a clock cycle of 3 time units, then platform A feeds time inputs from the set $\{0, 2, 4, 6, \ldots\}$ into S while platform B feeds time inputs from the set $\{0, 3, 6, 9, \ldots\}$ into S. Treating time as the input also has the added benefit of rendering S nonblocking and deterministic as an immediate consequence of existence and uniqueness of solutions for smooth differential equations. In Chapter 7 and Chapter 10 we discuss different ways in which dynamical systems can be modeled as systems.

Control systems

The national income model described in the previous section can be regarded as a control system if one assumes that the government expenditure at time $n+1$ can be chosen by the government. In this case we obtain the following equations:

$$\begin{aligned} c(n+1) &= \alpha\big(c(n) + i(n) + g(n)\big) \\ i(n+1) &= \beta\alpha\big(c(n) + i(n) + g(n)\big) - \beta c(n) \\ g(n+1) &= u(n) \end{aligned}$$

where u is regarded as an input to be chosen by the government and belonging to the set $[0, D]$ for some maximal expenditure $D \in \mathbb{R}^+$. This model defines a system with:

- $X = (\mathbb{R}_0^+)^3$;
- $X_0 = X$;
- $U = [0, D]$;
- $(c, i, g) \xrightarrow{u} (c', i', g')$ iff $c' = \alpha(c + i + g)$, $i' = \beta\alpha(c + i + g) - \beta c$, and $g' = u$;
- $Y = \mathbb{R}_0^+$;
- $H(c, i, g) = c + i + g$.

Control systems in continuous-time can also be described as systems. Consider the motion of a satellite subject to the torque produced by gas jet actuators. Its dynamics can be described by the forced version of Euler's equations:

$$\frac{d}{dt}\xi_1 = \frac{I_2 - I_3}{I_1}\xi_2\xi_3 + \upsilon_1 \tag{1.11}$$

$$\frac{d}{dt}\xi_2 = \frac{I_3 - I_1}{I_2}\xi_3\xi_1 + \upsilon_2 \tag{1.12}$$

$$\frac{d}{dt}\xi_3 = \frac{I_1 - I_2}{I_3}\xi_1\xi_2 \tag{1.13}$$

where the parameters υ_1 and υ_2 are inputs describing the effect of the gas jets. A system model for the controlled satellite requires a choice of curves $\mathcal{U}$, from $]a, b[$ to $\mathbb{R}^2$, to be used as inputs. From a mathematical point of view, we need to restrict $\mathcal{U}$ to a class of functions that are regular enough to guarantee existence and uniqueness of solutions for the differential equations (1.11) through (1.13). From an engineering point of view, the curves in $\mathcal{U}$ describe signals that are implemented by physical actuators. Therefore, there are limitations on the curves in $\mathcal{U}$ stemming from the actuator technology. Once the

set of input curves $\mathcal{U}$ is chosen, we can describe the dynamics of the controlled satellite by the system S with:

- $X = \mathbb{R}^3$;
- $X_0 = X$;
- $U = \mathcal{U}$;
- $x \xrightarrow{\upsilon} x'$ if there exist curves[2] $\mathcal{U} \ni \upsilon : [0, \tau] \to \mathbb{R}^2$ and $\xi : [0, \tau] \to \mathbb{R}^3$ satisfying equations (1.11) through (1.13) with $\xi(0) = x$ and $\xi(\tau) = x'$;
- $Y = X$;
- $H = 1_X$.

The sedulous reader certainly noticed that system S is also driven by time. An input $\upsilon \in \mathcal{U}$ is a curve $\upsilon : [0, \tau] \to \mathbb{R}^2$ describing not only the contribution of the gas jets to the angular acceleration, but also the *length of time* τ during which such contribution lasts. In particular, we can recover equations (1.8) through (1.10) from equations (1.11) through (1.13) by restricting $\mathcal{U}$ to be the set of all input curves with codomain $\{(0, 0)\} \subset \mathbb{R}^2$. In such case, each input $\upsilon \in \mathcal{U}$ is a constant curve assuming the value $(0, 0)$ on its domain $[0, \tau]$. Hence, we can identify υ with the length τ of its domain, thereby identifying $\mathcal{U}$ with $\mathbb{R}_0^+$, and recovering time as the input of system S modeling the rigid body without gas jets. Different system models for control systems appear in Chapter 8 and in Chapter 11.

1.3.3 Hybrid systems

Hybrid systems combine the characteristics of finite-state systems with those of infinite-state systems. We introduce them through examples.

Real-time scheduling

In many safety-critical applications the execution of software tasks needs to be completed in a timely fashion. Timeliness is typically formulated by equipping each software task with a worst case execution time C, a relative deadline D, and, in the case of periodic tasks, a period T. This formulation entails that a task becomes active every T units of time and its execution must finish no later than D units of time after becoming active. A scheduler is a special task that decides which of the active tasks should be executed by the processor. The decision process takes into account the period, relative deadline, and the fact that the execution may take, in the worst case, C units of time to complete. A set of tasks is said to be schedulable if it can be executed without violating any of the deadlines. We can model a periodic task by the hybrid system in Figure 1.6 where we assume that $C < D < T$ so that the task execution

[2] We use curves υ and ξ defined on closed sets while implicitly assuming the existence of curves $\upsilon' :]a, b[\to \mathbb{R}^2$ and $\xi' :]a, b[\to \mathbb{R}^3$ satisfying $\upsilon = \upsilon'|_{[0,\tau]}$, $\xi = \xi'|_{[0,\tau]}$, and all the additional conditions imposed on υ and ξ.

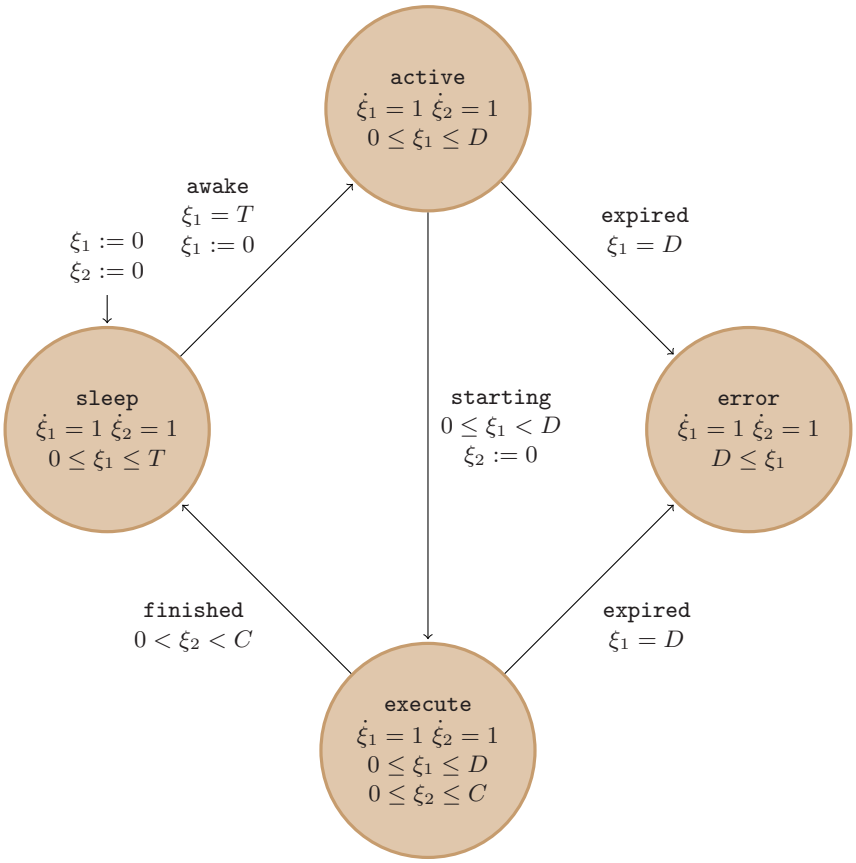

Fig. 1.6. Hybrid system representing a periodic real-time task.

can finish before the relative deadline and before the next activation. We also assume that the task cannot be pre-empted, *i.e.*, once the execution starts the task runs to completion.

There are several new ingredients in Figure 1.6 with respect to the finite-state systems that appeared so far. Firstly, each finite state is decorated with a differential equation which, in this example, is the same for all finite states:

$$\dot{\xi}_1 = 1 \tag{1.14}$$

$$\dot{\xi}_2 = 1. \tag{1.15}$$

The special form of this equation allows us to regard ξ_1 and ξ_2 as clocks measuring the passage of time since $\xi_1(t) = \xi_1(0) + t$ and $\xi_2(t) = \xi_2(0) + t$. Secondly, each finite state is also decorated with an *invariant* set. For example, the finite state `sleep` has $0 \le \xi_1 \le T$ as its invariant set. The invariant represents a condition on the solutions of the differential equation that must

be satisfied. This means that a task is in the finite state `sleep` provided that ξ_1 is between 0 and T. When ξ_1 reaches T, the invariant can no longer be satisfied since ξ_1 continues to increase according to $\dot{\xi}_1 = 1$. Consequently, a transition must be taken and the only transition originating at the finite state `sleep` is guarded by the condition $\xi_1 = T$. A guarded transition can only be taken when the *guard* is satisfied. Hence, the transition from the state `sleep` to the state `active` can only be taken when ξ_1 is exactly equal to T (the activation period). The *reset* $\xi_1 := 0$ also decorates this transition and forces ξ_1 to be reset to zero once the task enters the finite state `active`. It is also possible that a transition takes place at different instants of time. This is the case with the transition from `execute` to `sleep`. This transition is enabled whenever the guard $0 < \xi_2 < C$ is satisfied. Therefore, a task may finish its execution to enter the `sleep` state at any time, measured by ξ_2, between 0 and the worst case execution time C. The reader is encouraged to inspect the remaining invariants, guards, and resets to become convinced that the system in Figure 1.6 does model the evolution of a periodic real-time task.

The set of states of the hybrid system in Figure 1.6 is:

$$X = \{\texttt{sleep}, \texttt{active}, \texttt{error}, \texttt{execute}\} \times (\mathbb{R}_0^+)^2.$$

A state $x \in X$ is thus a pair $x = (x_a, x_b)$ consisting of a finite part $x_a \in \{\texttt{sleep}, \texttt{active}, \texttt{error}, \texttt{execute}\}$ and an infinite part $x_b \in (\mathbb{R}_0^+)^2$. For every finite state x_a we have an invariant $\mathrm{In}_{x_a} \subseteq (\mathbb{R}_0^+)^2$ and a differential equation $\dot{\xi} = f_{x_a}(\xi)$. Moreover, for every transition (x_a, u_a, x'_a) we have a guard $\mathrm{Gu}_{(x_a,u_a,x'_a)} \subseteq \mathrm{In}_{x_a}$ and a reset map $\mathrm{Re}_{(x_a,u_a,x'_a)} : \mathrm{In}_{x_a} \to \mathrm{In}_{x'_a}$. All these new ingredients are used to describe this hybrid system as a system $S = (X, X_0, U, \longrightarrow)$ with:

- $X = \{\texttt{sleep}, \texttt{active}, \texttt{error}, \texttt{execute}\} \times (\mathbb{R}_0^+)^2$;
- $X_0 = \{(x_a, x_b) \in X \mid x_a = \texttt{sleep} \ \wedge \ x_b \in \mathrm{In}_{\texttt{sleep}}\}$;
- $U = \{\texttt{awake}, \texttt{expired}, \texttt{starting}, \texttt{finished}\} \cup \mathbb{R}_0^+$;
- $(x_a, x_b) \xrightarrow{u} (x'_a, x'_b)$ if one of the following two conditions holds:
  1. $u \in \mathbb{R}_0^+$, $x'_a = x_a$, and there exists a solution $\xi : [0, u] \to \mathrm{In}_{x_a}$ to the differential equations (1.14) and (1.15) satisfying $\xi(0) = x_b$ and $\xi(u) = x'_b$;
  2. $u \in \{\texttt{awake}, \texttt{expired}, \texttt{starting}, \texttt{finished}\}$, $x'_b = \mathrm{Re}_{(x_a,u,x'_a)}(x_b)$, and $x_b \in \mathrm{Gu}_{(x_a,u,x'_a)}$.

The transition relation consists of two different kinds of transitions: continuous flows and discrete transitions. During continuous flows the finite part of the state remains unaltered and the infinite part of the state, whose evolution is prescribed by the differential equations (1.14) and (1.15), remains inside the invariant set. Discrete transitions may change the finite and the infinite part of the state and are conditioned by the corresponding guards.

A boost DC-DC converter

A different example of a hybrid system is the boost DC-DC converter represented in Figure 1.7. If one uses the current i_L through the inductor and the voltage v_C across the capacitor as state variables, a simple application of Kirchoff's laws provides the equations describing the evolution of v_C and i_L. When the switch is in position s_1 we have:

$$\frac{d}{dt}i_L = -\frac{R_L}{L}i_L + \frac{1}{L}v_S \tag{1.16}$$

$$\frac{d}{dt}v_C = -\frac{1}{C}\frac{1}{R_C + R_0}v_C \tag{1.17}$$

and when the switch is in position s_2:

$$\frac{d}{dt}i_L = -\frac{1}{L}\left(R_L + \frac{R_C R_0}{R_C + R_0}\right)i_L - \frac{1}{L}\frac{R_0}{R_C + R_0}v_C + \frac{1}{L}v_S \tag{1.18}$$

$$\frac{d}{dt}v_C = \frac{1}{C}\frac{R_0}{R_C + R_0}i_L - \frac{1}{C}\frac{1}{R_C + R_0}v_C. \tag{1.19}$$

There are two different modes of operation associated with the two different positions for the switch. The dynamics of these two modes of operation can be described by the finite-state system in Figure 1.8 in which the input u_1 takes the system to switch position s_1 and input u_2 takes the system to switch position s_2. However, the finite-state system in Figure 1.8 is not detailed enough to capture the dynamics of i_L and v_C. A more detailed model

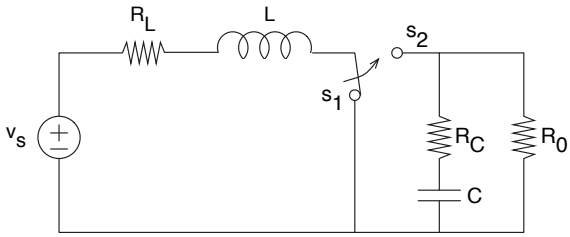

Fig. 1.7. Boost DC-DC converter.

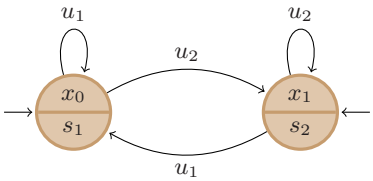

Fig. 1.8. Finite-state system describing the dynamics of the switch in the converter represented in Figure 1.7.

has to combine the finite-state dynamics of the switch with the continuous-time dynamics in equations (1.16) through (1.19). Such hybrid model can be described by the system $S = (X, U, \longrightarrow)$ with:

- $X = \{s_1, s_2\} \times \mathbb{R}^2$;
- $U = \{u_1, u_2\} \cup \mathbb{R}_0^+$;
- $(s, i_L, v_C) \xrightarrow{u} (s', i'_L, v'_C)$ if one of the following four conditions holds:
  1. $u = u_1$, $s' = s_1$, $i_L = i'_L$, and $v_C = v'_C$;
  2. $u = u_2$, $s' = s_2$, $i_L = i'_L$, and $v_C = v'_C$;
  3. $u \in \mathbb{R}_0^+$, $s' = s = s_1$ and there exists a solution $\xi : [0, u] \to \mathbb{R}^2$ to the differential equations (1.16) and (1.17) satisfying $\xi(0) = (i_L, v_C)$ and $\xi(u) = (i'_L, v'_C)$;
  4. $u \in \mathbb{R}_0^+$, $s' = s = s_2$ and there exists a solution $\xi : [0, u] \to \mathbb{R}^2$ to the differential equations (1.18) and (1.19) satisfying $\xi(0) = (i_L, v_C)$ and $\xi(u) = (i'_L, v'_C)$.

For this hybrid system the invariants are $\mathrm{In}_{s_1} = \mathbb{R}^2 = \mathrm{In}_{s_2}$, the guards are $\mathrm{Gu}_{(s_1,u,s_2)} = \mathbb{R}^2 = \mathrm{Gu}_{(s_2,u,s_1)}$, and the reset maps are $\mathrm{Re}_{(s_1,u,s_2)} = 1_{\mathbb{R}^2} = \mathrm{Re}_{(s_2,u,s_1)}$ for any $u \in \{u_1, u_2\}$. The nature of the guards and of the invariants implies that a discrete transition can take place independently of the specific value of the infinite state. This is in stark contrast with the previous example where discrete transitions were influenced by and influenced the continuous-time dynamics. The modeling of hybrid systems as systems is discussed in more detail in Chapter 7.

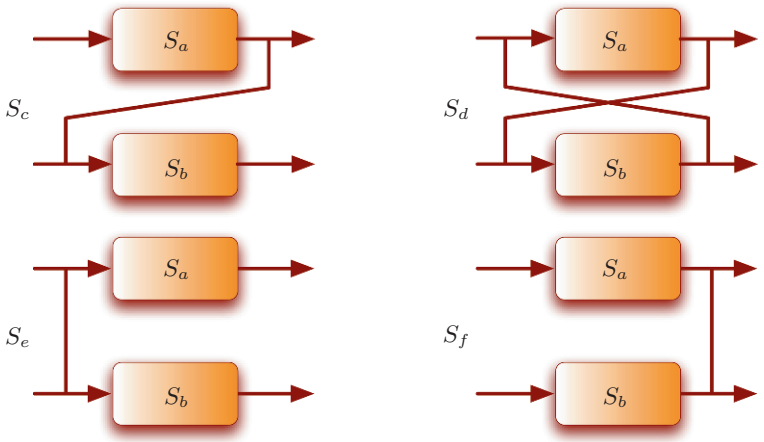

Fig. 1.9. Systems S_c, S_d, S_e, and S_f resulting from composing system S_a with system S_b with respect to the different interconnection relations in Table 1.1.

1.4 Composing systems

Most engineering systems are designed and built by interconnecting simpler and smaller components. This process of constructing larger systems by interconnecting smaller ones can be mathematically described through a composition operation. In this book we consider a versatile notion of composition based on an *interconnection relation* $\mathcal{I} \subseteq X_a \times X_b \times U_a \times U_b$ describing how system S_a interacts with system S_b.

Definition 1.6 (Composition). *Let* $S_a = (X_a, X_{a0}, U_a, \xrightarrow[a]{}, Y_a, H_a)$ *and* $S_b = (X_b, X_{b0}, U_b, \xrightarrow[b]{}, Y_b, H_b)$ *be two systems and let* $\mathcal{I} \subseteq X_a \times X_b \times U_a \times U_b$ *be a relation. The* composition *of* S_a *and* S_b *with interconnection relation* $\mathcal{I}$*, denoted by* $S_a \times_{\mathcal{I}} S_b$*, is the system* $(X_{ab}, X_{ab0}, U_{ab}, \xrightarrow[ab]{}, Y_{ab}, H_{ab})$ *consisting of:*

- $X_{ab} = \pi_X(\mathcal{I})$;
- $X_{ab0} = X_{ab} \cap (X_{a0} \times X_{b0})$;
- $U_{ab} = U_a \times U_b$;
- $(x_a, x_b) \xrightarrow[ab]{(u_a, u_b)} (x'_a, x'_b)$ *if the following three conditions hold:*
  1. $x_a \xrightarrow[a]{u_a} x'_a$ *in* S_a;
  2. $x_b \xrightarrow[b]{u_b} x'_b$ *in* S_b;
  3. $(x_a, x_b, u_a, u_b) \in \mathcal{I}$;
- $Y_{ab} = Y_a \times Y_b$;
- $H_{ab}(x_a, x_b) = (H_a(x_a), H_b(x_b))$.

The system $S_a \times_{\mathcal{I}} S_b$ describes the concurrent evolution of systems S_a and S_b subject to the synchronization prescribed by the interconnection relation $\mathcal{I}$. The compositions represented in Figure 1.9 are but a few examples of systems that can be obtained by suitably defining $\mathcal{I}$. The interconnection relation for these examples is shown in Table 1.1. The very general notion of composition in Definition 1.6 was not chosen for the mere sake of generality. The effort placed by the reader in becoming acquainted with this notion of composition will be rewarded later in the book with simple proofs of important results.

System	$(x_a, x_b, u_a, u_b) \in \mathcal{I}$ if
S_c	$H_a(x_a) = u_b$
S_d	$H_a(x_a) = u_b$ and $H_b(x_b) = u_b$
S_e	$u_a = u_b$
S_f	$H_a(x_a) = H_b(x_b)$

Table 1.1. Description of the interconnection relation $\mathcal{I}$ for the compositions represented in Figure 1.9.

The compositions illustrated in Figure 1.9 enforce synchronization only through inputs and outputs. However, the relation $\mathcal{I}$ enables also the use of the state for synchronization. This will be essential when we discuss problems of control. The way in which transitions of $S_a \times_{\mathcal{I}} S_b$ are constructed from transitions of S_a and S_b tells us that:

$$\mathcal{B}(S_a \times_{\mathcal{I}} S_b) \subseteq \mathcal{B}(S_a) \times \mathcal{B}(S_b) \tag{1.20}$$

with equality holding for the trivial interconnection relation defined by $\mathcal{I} = X_a \times X_b \times U_a \times U_b$. In the later case, we denote $S_a \times_{\mathcal{I}} S_b$ simply by $S_a \times S_b$.

When composing systems S_a and S_b having the same set of outputs and using an interconnection relation satisfying:

$$(x_a, x_b) \in \pi_X(\mathcal{I}) \implies H_a(x_a) = H_b(x_b)$$

there is a certain redundancy in $S_a \times_{\mathcal{I}} S_b$. The output set is $Y_a \times Y_b$ with $Y_a = Y_b$ and the output at every state $(x_a, x_b) \in X_{ab}$ is $(H_a(x_a), H_b(x_b))$ with $H_a(x_a) = H_b(x_b)$. Under these circumstances we simplify $S_a \times_{\mathcal{I}} S_b$ by redefining Y_{ab} to be:

$$Y_{ab} = Y_a = Y_b$$

and by redefining H_{ab} to be:

$$H_{ab}(x_a, x_b) = H_a(x_a) = H_b(x_b).$$

Equation (1.20) now tells us that the behavior of $S_a \times_{\mathcal{I}} S_b$ is related to be behavior of S_a and S_b by:

$$\mathcal{B}(S_a \times_{\mathcal{I}} S_b) \subseteq \mathcal{B}(S_a) \qquad \mathcal{B}(S_a \times_{\mathcal{I}} S_b) \subseteq \mathcal{B}(S_b).$$

Example 1.7. We now return to the communication protocol example to illustrate system composition. If S_a is the system describing the sender (see Figure 1.3) and if S_b is the system describing the receiver (see Figure 1.4), we can construct $S_a \times_{\mathcal{I}} S_b$ by choosing $\mathcal{I}$ to be the set of all quadruples $(x_a, x_b, u_a, u_b) \in X_a \times X_b \times U_a \times U_b$ satisfying $H_a(x_a) = H_b(x_b)$ and corresponding to one of the interconnections represented in Figure 1.9. The resulting finite-state system is represented in Figure 1.10.

A possible specification for the communication protocol is that in every external behavior of $S_a \times_{\mathcal{I}} S_b$, `tx/rx` is always followed, although not necessarily immediately, by `ack`. This would require that any transmission is eventually correctly received. By inspecting Figure 1.10, we see that it is possible to cycle between the states `(send data, receive data)` and `(bad tx, bad rx)` indefinitely, thus implying that `tx/rx` may not be followed by `ack`. Unless additional assumptions are made about the communication channel, it is not possible to guarantee that transmitted messages are eventually received. In this example we could determine that $S_a \times_{\mathcal{I}} S_b$ does not conform to the specification simply by inspecting $S_a \times_{\mathcal{I}} S_b$. When dealing with large systems, analyzing $S_a \times_{\mathcal{I}} S_b$ by inspection is no longer possible and one has to resort to algorithmic verification techniques such as the ones presented in Part II. $\lhd$

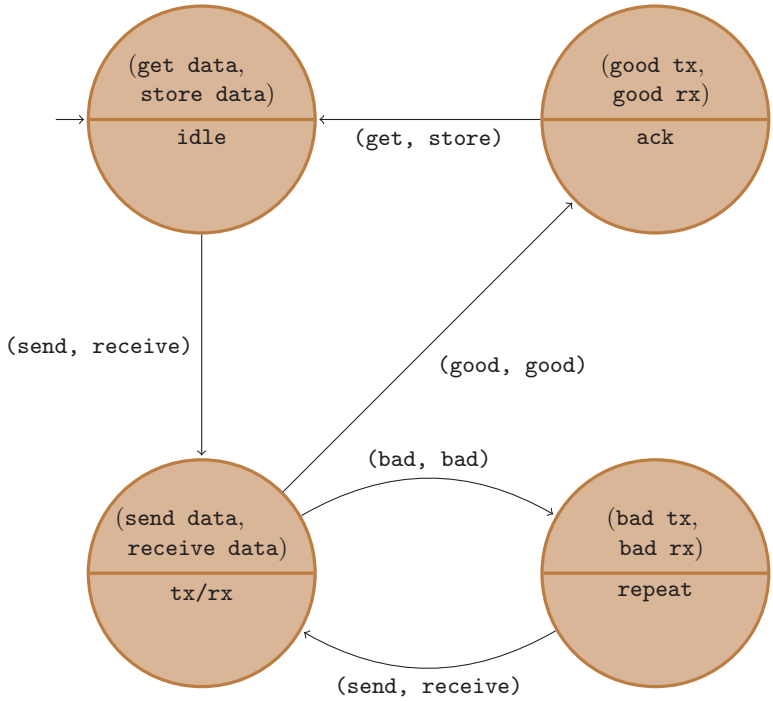

Fig. 1.10. Graphical representation of the composition of the finite-state system displayed in Figure 1.3 (modeling the sender) with the finite-state system displayed in Figure 1.4 (modeling the receiver).

1.5 Notes

The notion of system introduced in this chapter is a blend between the automata theoretic models used for the verification of software systems with the differential equation models used in control. Owing to its automata descent, the dynamics is described by a relation instead of a function. The flexibility afforded by relations will be used in Parts II, III, and IV to model the effect of disturbances through nondeterminism. Following the current practice in control theory, outputs depend on the states rather than on the inputs. This choice is motivated by the models used in control theory where the state has physical meaning which is essential to define specifications. However, as described in Section 1.2, the inputs can easily be extracted from the outputs by suitable redefining the ingredients of a system.

The separation of variables into inputs, states, and outputs is a modeling decision that is not always easy to make, especially when dealing with software. Although this practice seems to be well rooted in the computer science and control theory traditions, there exist alternative modeling formalisms that do not require such distinctions [PW97].

The national income model was proposed in [Sam39]. Hybrid systems have been used by many authors to study problems of real-time scheduling see, for example, [FKPY07]. The hybrid model for the DC-DC converter is taken from [GPM04, GPT09] where further references can be found for the analysis and design of DC-DC converters using hybrid systems techniques.