# Chapter 1
# Modelling of Engineering Phenomena by Finite Automata

Jörg Raisch

## 1.1 Introduction

In "conventional" systems and control theory, signals "live" in $\mathbb{R}^n$ (or some other, possibly infinite-dimensional, vector space). Then, a signal is a map $T \to \mathbb{R}^n$, where $T$ represents continuous or discrete time. There are, however, numerous application domains where signals only take values in a discrete set, which is often finite and not endowed with mathematical structure. Examples are pedestrian lights (possible signal values are "red" and "green") or the qualitative state of a machine ("busy", "idle", "down"). Often, such signals can be thought of as naturally discrete-valued; sometimes, they represent convenient abstractions of continuous-valued signals and result from a quantisation process.

**Example 1.1.** Consider a water reservoir, where $z : \mathbb{R}^+ \to \mathbb{R}^+$ is the (continuous-valued) signal representing the water level in the reservoir. The quantised signal

$$\tilde{y}: \quad \mathbb{R}^+ \to \{\texttt{Hi}, \texttt{Med}, \texttt{Lo}\} \ ,$$

where

$$\tilde{y}(t) = \begin{cases} \texttt{Hi} \text{ if } & z(t) > 2 \\ \texttt{Med} \text{ if } & 1 < z(t) \leq 2 \\ \texttt{Lo} \text{ if } & z(t) \leq 1 \end{cases}$$

represents coarser, but often adequate, information on the temporal evolution of the water level within the reservoir. This is indicated in Fig. 1.1, which also shows that the discrete-valued signal $\tilde{y}$ can be represented by a sequence or string of timed discrete events, e.g.,

$$(t_0, \texttt{Lo}), (t_1, \texttt{Med}), (t_2, \texttt{Hi}), \ldots ,$$

Jörg Raisch
Fachgebiet Regelungssysteme, TU Berlin, Sekr. EN11, Einsteinufer 17,
10587 Berlin, Germany. Also: Fachgruppe System- und Regelungstheorie,
Max-Planck-Institut für Dynamik komplexer technischer Systeme, Magdeburg
e-mail: raisch@control.tu-berlin.de

C. Seatzu et al. (Eds.): Control of Discrete-Event Systems, LNCIS 433, pp. 3–22.
springerlink.com 

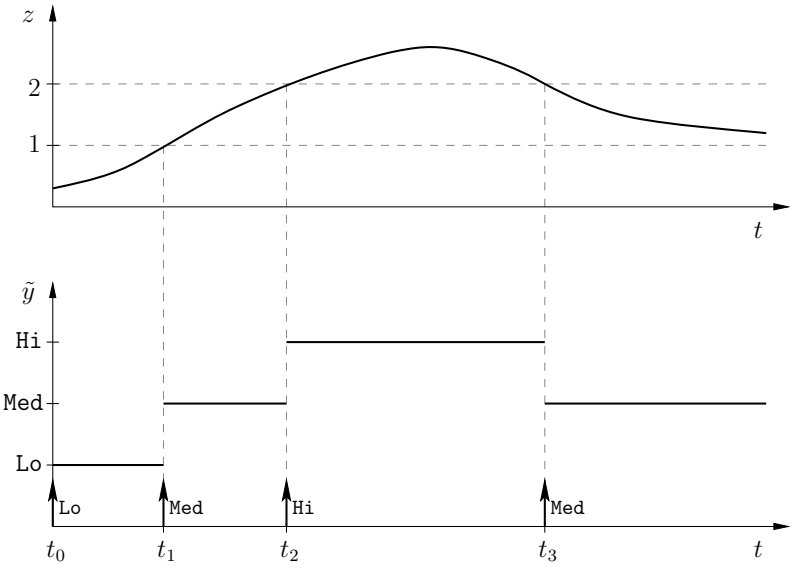


**Fig. 1.1** Quantisation of a continuous signal

where $t_i \in \mathbb{R}^+$ are event times and $(t_i, \texttt{Med})$ means that at time $t_i$, the quantised signal $\tilde{y}$ changes its value to `Med`. ■

Note that an (infinite) sequence of timed discrete events can be interpreted as a map $\mathbb{N}_0 \to \mathbb{R}^+ \times Y$, where $Y$ is an event set. Similarly, a finite string of timed discrete events can be seen as a map defined on an appropriate finite subset $I_j = \{0, \ldots, j\}$ of $\mathbb{N}_0$.

Often, even less information may be required. For example, only the temporal ordering of events, but not the precise time of the occurrence of events may be relevant. In this case, we can simply project out the time information and obtain a sequence (or string) of logical events, e.g.,

$$\texttt{Lo}, \texttt{Med}, \texttt{Hi}, \ldots,$$

which can be interpreted as a map $y : \mathbb{N}_0$ (resp. $I_j$) $\to Y$, where $Y$ is the event set. It is obvious (but important) to note, that the domain $\mathbb{N}_0$ (respectively $I_j$) does in general *not* represent uniformly sampled time; i.e., the time difference $t_{k+1} - t_k$, with $k, k+1 \in \mathbb{N}_0$ (respectively $I_j$), between the occurrence of subsequent events $y(k+1)$ and $y(k)$ is usually not a constant.

Clearly, going from the continuous-valued signal $z$ to the discrete-valued signal $\tilde{y}$ (or the corresponding sequence of timed discrete events), and from the latter to a sequence $y$ of logical events, involves a loss of information. This is often referred to as signal aggregation.

We will use the following terminology: given a set $U$, the symbol $U^{\mathbb{N}_0}$ denotes the set of all functions mapping $\mathbb{N}_0$ into $U$, i.e., the set of all (infinite) sequences of elements from $U$. The symbol $U^*$ denotes the set of all finite strings from elements of $U$. More specifically, $U^{I_j} \subset U^*$, $j = 0, 1, \ldots$, is the set of strings from $U$ with length $j+1$, and $\varepsilon$ is the empty string. The length of a string $u$ is denoted by $|u|$, i.e., $|u| = j+1$ if $u \in U^{I_j}$, and $|\varepsilon| = 0$.

We will be concerned with models that explain sequences or strings of logical discrete events. In all but trivial cases, these models will be dynamic, i.e., to explain the $k$-th event, we will need to consider previous events. As in "conventional" systems and control theory, it is convenient to concentrate on state models.

## 1.2 State Models with Inputs and Outputs

The traditional control engineering point of view is based on the following assumptions (compare Fig. 1.2):

- the system to be controlled (plant) exhibits input and output signals;
- the control input is free, i.e., it can be chosen by the controller;
- the disturbance is an input that cannot be influenced; in general, it can also not be measured directly;
- the output can only be affected indirectly;
- the plant model relates control input and output signals;
- the disturbance input may or may not be modelled explicitly; if it is not, the existence of disturbances simply "increases the amount of nondeterminism" in the (control) input/output model;
- the design specifications depend on the output; they may additionally depend on the control input.

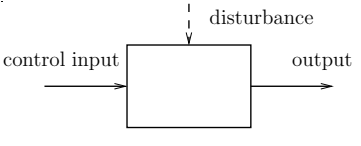


**Fig. 1.2** System with inputs and outputs

### *1.2.1 Mealy Automata*

The relation between discrete-valued input and output signals can often be modelled by finite Mealy automata:

**Definition 1.1.** *A deterministic finite Mealy automaton (DFMeA) is a sixtuple* $(X,U,Y,f,g,x_0)$*, where*

- $X = \{\xi_1, \dots \xi_n\}$ *is a finite set of states,*
- $U = \{\mu_1, \dots \mu_q\}$ *is a finite set of input symbols,*
- $Y = \{\nu_1, \dots \nu_p\}$ *is a finite set of output symbols,*
- $f : X \times U \to X$ *represents the transition function,*
- $g : X \times U \to Y$ *represents the output function,*
- $x_0 \in X$ *is the initial state.*

Note that the transition function is a total function, i.e., it is defined for all pairs in $X \times U$. In other words, we can apply any input symbol in any state, and the input signal $u$ is therefore indeed free. Furthermore, as $x_o$ is a singleton and the transition function maps into $X$, any sequence (string) of input symbols will provide a unique sequence (string) of output symbols. Finally, from the state and output equations

$$x(k+1) = f(x(k),u(k)), \tag{1.1}$$
$$y(k) = g(x(k),u(k)), \tag{1.2}$$

it is obvious that the output symbols $y(0) \dots y(k-1)$ will not depend on the input symbols $u(k),u(k+1),\dots$. One says that the output *does not anticipate* the input, or, equivalently, that the input output relation is non-anticipating [19].

**Remark 1.2.** Sometimes, it proves convenient to extend the definition of DFMeA by adding a set $X_m \subseteq X$, the set of marked states. The role of $X_m$ is to model acceptable outcomes when applying a finite string of input symbols. Such an outcome is deemed to be achieved when a string of input symbols "drives" the system state from $x_0$ into $X_m$. One could argue, of course, whether the definition of an acceptable outcome should be contained in the plant model or rather be part of the specification, which is to be modelled separately. ■

The following example illustrates how DFMeA can model discrete event systems with inputs and outputs.

**Example 1.3.** Let us consider a candy machine with (very) restricted functionality. It sells two kinds of candy, chocolate bars (product 1) and chewing gum (product 2). A chocolate bar costs 1 €, chewing gum costs 2 €. The machine only accepts 1 € and 2 € coins and will not give change. Finally, we assume that the machine is sufficiently often refilled and therefore never runs out of chocolate or chewing gum. In this simplified scenario, the customer can choose between the input symbols:

| | |
|---|---|
| 1 € | "insert a 1 € coin" |
| 2 € | "insert a 2 € coin" |
| P1 | "request product 1 (chocolate bar)" |
| P2 | "request product 2 (chewing gum)" |

The machine can respond with the following output symbols:

G1 "deliver product 1"
G2 "deliver product 2"
M1 "display message `insufficient credit`"
M2 "display message `choose a product`"
R1€ "return 1€ coin"
R2€ "return 2€ coin"

Clearly, to implement the desired output response, the machine needs to keep track of the customer's current credit. To keep the system as simple as possible, we restrict the latter to 2€. Hence, we will have the following states:

$\xi_1$ "customer's credit is 0€",
$\xi_2$ "customer's credit is 1€",
$\xi_3$ "customer's credit is 2€",

where $\xi_1$ is the initial state and the only marked state, i.e., $x_0 = \xi_1$ and $X_m = \{\xi_1\}$. The Mealy automaton shown in Fig. 1.3 models the functionality of our simple candy machine. In this figure, circles represent states, and arrows represent transitions between states. Transitions are labelled by a pair $\mu_i/\nu_j$ of input/output symbols. For example, the arrow labelled 2€/M2 beginning in $\xi_1$ and ending in $\xi_3$ is to be interpreted as $f(\xi_1, 2€) = \xi_3$ and $g(\xi_1, 2€) = \text{M2}$. Hence, if the customer's credit is 0€ and (s)he inserts a 2€ coin, (s)he will subsequently have 2€ credit, and the machine will display the message `choose a product`. The initial state is indicated by a small arrow pointing towards it "from the outside", and a marked state is shown by a small arrow pointing from the state "towards the outside". ■

The following notions characterise how a DFMeA responds to sequences (strings) of input symbols:

The set of all pairs of input/output signals (or, equivalently, the set of all pairs of sequences of input and output symbols) which are compatible with the automaton

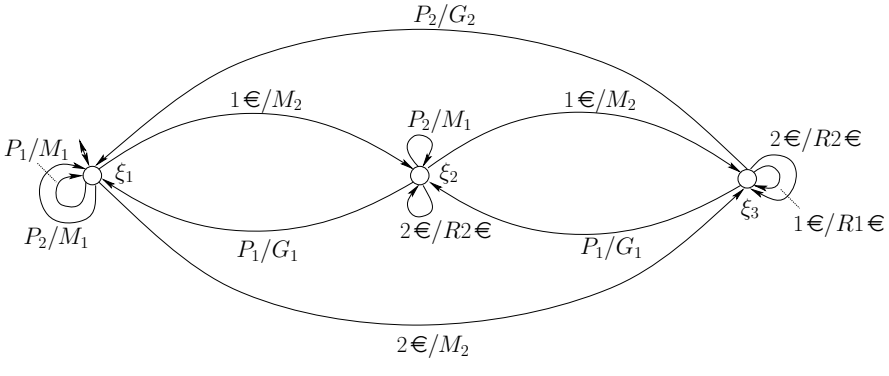


**Fig. 1.3** Mealy automaton modelling a candy machine

dynamics is called the *behaviour generated by a DFMeA* and denoted by $\mathscr{B}$. Formally,

$$\mathscr{B} = \left\{ (u,y) \mid \exists x \in X^{\mathbb{N}_0} \text{ s.t. (1.1) and (1.2) hold } \forall k \in \mathbb{N}_0,\, x(0) = x_0 \right\}.$$

Hence, a pair $(u,y)$ of input and output sequences is in the behaviour $\mathscr{B}$ if and only if there exists a state sequence $x$ that begins in the initial state and, together with $(u,y)$, satisfies the next state and output equations defined by $f$ and $g$.

The *language generated by a DFMeA*, denoted by $L$, is the set of all pairs $(u,y)$ of input and output strings with equal length such that applying the input string $u \in U^*$ in the initial automaton state $x_0$ makes the DFMeA respond with the output string $y \in Y^*$. Formally,

$$L = \Big\{ (u,y) \mid |u| = |y|,\, \exists x \in X^{I_{|u|}} \text{ s.t. (1.1) and (1.2) hold for } k = 0,\ldots,|u|-1,\, x(0) = x_0 \Big\}.$$

If a set of marked states is given, we can also define the *language marked by a DFMeA*, $L_m$. It represents the elements of the language $L$ that make the state end in $X_m$. Formally,

$$L_m = \{ (u,y) \mid (u,y) \in L,\, x(|u|) \in X_m \}.$$

Clearly, in our candy machine example, ((P1,P1,...), (M1, M1, ...)) $\in \mathscr{B}$, ((1€,P2),(M2,M1)) $\in L$, and ((1€,P1),(M2,G1)) $\in L_m$.

Modelling is in practice often intentionally coarse, i.e., one tries to model only phenomena that are important for a particular purpose (e.g., the synthesis of feedback control). In the context of input/output models, this implies that the effect of inputs on outputs is, to a certain extent, uncertain. This is captured by the notion of nondeterministic finite Mealy automata (NDFMeA).

**Definition 1.2.** *A nondeterministic finite Mealy automaton (NDFMeA) is a quintuple $(X,U,Y,h,X_0)$, where*

- *$X$ is a finite set of states,*
- *$U$ is a finite set of input symbols,*
- *$Y$ is a finite set of output symbols,*
- *$h : X \times U \to 2^{X \times Y} \setminus \emptyset$ represents a set-valued transition-output function,*
- *$X_0 \subseteq X$, $X_0 \neq \emptyset$ is a set of possible initial states.*

Hence, there are two "sources" of uncertainty in NDFMeA: (i) the initial state may not be uniquely determined; (ii) if the automaton is in a certain state and an input symbol is applied, the next state and/or the generated output symbol may be nondeterministic. The evolution is now governed by

$$(x(k+1), y(k)) \in h(x(k), u(k)). \tag{1.3}$$

As in the deterministic case, the output symbols $y(0)\ldots y(k-1)$ will not depend on the input symbols $u(k), u(k+1), \ldots$, i.e., the output does not anticipate the input.

**Example 1.4.** Consider the slightly modified model of our candy machine in Fig. 1.4, which includes a limited "change" policy. The only difference compared to Example 1.3 is when the machine is in state $\xi_2$ (i.e., the customer's credit is 1 €), and a 2 € coin is inserted. Depending on whether 1 € coins are available (which is not modelled), the machine may return a 1 € coin and go to state $\xi_3$ (i.e., the customer's credit is now 2 €), or it may return the inserted 2 € coin and remain in state $\xi_2$. ■

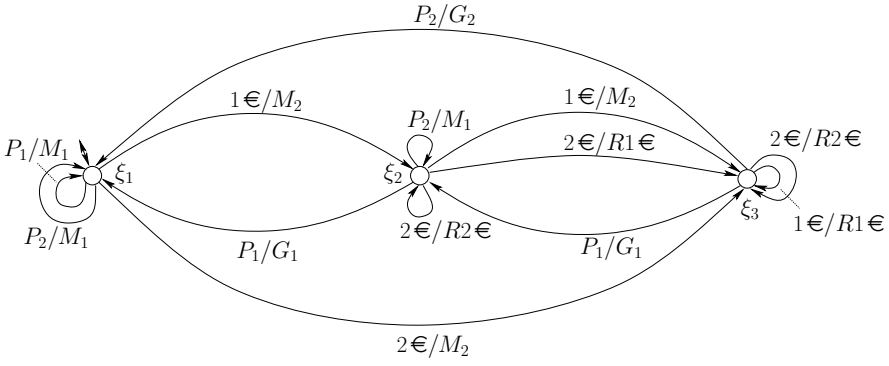


**Fig. 1.4** Nondeterministic Mealy automaton modelling a candy machine

**Remark 1.5.** Note that a combined transition-output function $h : X \times U \to 2^{X\times Y} \setminus \emptyset$ is more expressive in the set-valued case than separate transition and output functions $f : X \times U \to 2^X \setminus \emptyset$ and $g : X \times U \to 2^Y \setminus \emptyset$. This is illustrated by Example 1.4 above. There, $h(\xi_2, 2€) = \{(\xi_3, \mathrm{R}1€), (\xi_2, \mathrm{R}2€)\}$. That is, in state $\xi_2$, after choosing input 2 €, there are two possibilities for the next state and two possibilities for the resulting output symbol; however, arbitrary combinations of these are not allowed. This could not be modelled by separate transition and output functions $f, g$. ■

**Remark 1.6.** The use of a combined transition-output function also leads to a more compact formulation of the behaviour and the language generated by an NDFMeA:

$$\mathscr{B} = \left\{(u,y) \;\middle|\; u \in U^{\mathbb{N}_0}, y \in Y^{\mathbb{N}_0},\ \exists x \in X^{\mathbb{N}_0} \text{ s.t. (1.3) holds } \forall k \in \mathbb{N}_0,\ x(0) \in X_0\right\},$$

$$L = \Big\{(u,y) \mid u \in U^*, y \in Y^*, |u| = |y|,\ \exists x \in X^{I_{|u|}} \text{ s.t. (1.3) holds for } k = 0,\dots,|u|-1,\ x(0) \in X_0\Big\}.$$ ■

**Remark 1.7.** As in the deterministic case, we can extend the definition by adding a set of marked states, $X_m \subseteq X$. The language marked by an NDFMeA is then the subset of $L$ that admits a corresponding string of states to end in $X_m$, i.e.,

$$L_m = \Big\{(u,y) \mid u \in U^*, y \in Y^*, |u| = |y|,\ \exists x \in X^{I_{|u|}} \text{ s.t. (1.3) holds for } k = 0,\dots,|u|-1,\ x(0) \in X_0,\ x(|u|) \in X_m\Big\}.$$

Note that at least one string of states compatible with a $(u,y) \in L_m$ must end in $X_m$, but not all of them have to. ■

### 1.2.2 Moore Automata

Often, one encounters situations where the $k$th output event does not depend on the $k$th (and subsequent) input event(s). In this case, it is said that the output *strictly* does not anticipate the input. Clearly, this is achieved, if the output function of a DFMeA is restricted to $g : X \to Y$. The resulting state machine is called a Moore automaton:

**Definition 1.3.** *A deterministic finite Moore automaton (DFMoA) is a sixtuple* $(X,U,Y,f,g,x_0)$*, where*

- $X$ *is a finite set of states,*
- $U$ *is a finite set of input symbols,*
- $Y$ *is a finite set of output symbols,*
- $f : X \times U \to X$ *represents the transition function,*
- $g : X \to Y$ *represents the output function,*
- $x_0 \in X$ *is the initial state.*

The evolution of state and output is determined by

$$\begin{aligned} x(k+1) &= f(x(k),u(k)), \\ y(k) &= g(x(k)). \end{aligned}$$

**Remark 1.8.** As for Mealy automata, the above definition can be extended by adding a set of marked states, $X_m \subseteq X$, to model acceptable outcomes when applying a finite string of input symbols. ■

**Remark 1.9.** A nondeterministic version (NDFMoA) is obtained by the following straightforward changes in the above definition: the initial state is from a set $X_o \subseteq X$, $X_o \neq \emptyset$; the transition function and the output function map into the respective power sets, i.e., $f : X \times U \to 2^X \setminus \emptyset$ and $g : X \to 2^Y \setminus \emptyset$. ■

## 1.3 Automata with Controllable and Uncontrollable Events

Although distinguishing between control inputs and outputs is a natural way of modelling the interaction between plant and controller, in most of the work in the area of discrete event control systems a slightly different point of view has been adopted. There, plants are usually modelled as deterministic finite automata with an event set that is partitioned into sets of controllable (i.e., preventable by a controller) and uncontrollable (i.e., non-preventable) events. For example, starting your car is a

controllable event (easily prevented by not turning the ignition key), whereas breakdown of a car is, unfortunately, uncontrollable. In this scenario, a string or sequence of events is in general not an input and therefore not free. Hence, the transition function is usually defined to be a partial function. The resulting machine is often referred to as a finite generator.

**Definition 1.4.** *A deterministic finite generator (DFG) is a sixtuple* $(X, S, S_c, f, x_0, X_m)$*, where*

- $X = \{\xi_1, \dots \xi_n\}$ *is a finite set of states,*
- $S = \{\sigma_1 \dots \sigma_q\}$ *is a finite set of events (the "alphabet"),*
- $S_c \subseteq S$ *is the set of controllable events; consequently* $S_{uc} = S \setminus S_c$ *represents the set of uncontrollabe events,*
- $f : X \times S \to X$ *is a partial transition function,*
- $x_0 \in X$ *is the initial state,*
- $X_m \subseteq X$ *is a set of marked states.*

As for input/output automata, we distinguish the language generated and the language marked by DFG:

$$L = \Big\{ s \in S^* \mid \exists x \in X^{I_{|s|}} \text{ s.t. } x(k+1) = f(x(k), s(k)),\ k = 0, \dots, |s|-1,\ x(0) = x_0 \Big\},$$

$$L_m = \Big\{ s \in S^* \mid \exists x \in X^{I_{|s|}} \text{ s.t. } x(k+1) = f(x(k), s(k)),\ k = 0, \dots, |s|-1,\ x(0) = x_0,\ x(|s|) \in X_m \Big\}.$$

The set of events that can occur in state $\xi$ is called the active event set, denoted by $\Gamma(\xi)$, i.e.,

$$\Gamma(\xi) = \{\sigma \mid f \text{ is defined on } (\xi, \sigma)\}.$$

**Example 1.10.** Consider the following simple model of a simple machine (taken from [21]): the model has three states, `idle`, `working`, and `broken`. The event set $S$ consists of four elements:

| | |
|---|---|
| $a$ | `take a workpiece and start processing it,` |
| $b$ | `finish processing of workpiece,` |
| $c$ | `machine breaks down,` |
| $d$ | `machine gets repaired.` |

The events $a$ and $d$ are controllable in the sense that they can be prevented, i.e., $S_c = \{a, d\}$. It is assumed that neither the breakdown of the machine nor the finishing of a workpiece can be prevented by control, i.e., $S_{uc} = \{b, c\}$. The state `idle` is both the initial state and the only marked state. The transition structure is shown in Fig. 1.5, where, e.g., the arc from `idle` to `working` labelled by $a$ means that the transition function is defined for the pair $(\texttt{idle}, a)$ and $f(\texttt{idle}, a) = \texttt{working}$. To distinguish controllable and uncontrollable events, arcs labelled with controllable events are equipped with a small bar. Clearly, in this example, the strings *ab*, *aba*,

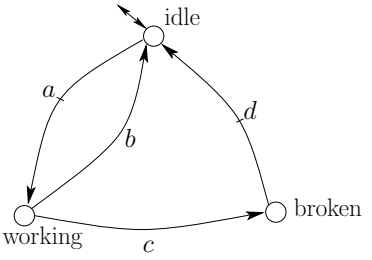


**Fig. 1.5** Simple machine model

and *acd* are in the language generated by the DFG in Fig. 1.5, and *ab*, *acd* (but not *aba*) are also in the marked language. ■

As indicated in the previous section, nondeterminism is in practice a feature that is often intentionally included to keep models simple.

**Definition 1.5.** *A nondeterministic finite generator (NDFG) is a sixtuple* $(X, S, S_c, f, X_0, X_m)$*, where*

- $X$*,* $S$*,* $S_c$ *and* $X_m$ *are as in Definition 1.4,*
- $f : X \times S \to 2^X$ *is the transition function,*
- $X_0 \subseteq X$ *is the set of possible initial states.*

Note that the transition function, as it maps into the power set $2^X$ (which includes the empty set), can be taken as a total function. That is, it is defined for all state-event pairs, and $\sigma \in \Gamma(\xi)$ if and only if $f(\xi, \sigma) \neq \emptyset$. The definitions of languages generated and marked by NDFGs carry over from the deterministic case with the obvious change that $x(k+1) = f(x(k), s(k))$ needs to be replaced by $x(k+1) \in f(x(k), s(k))$.

## 1.4 Finite Automata as Approximations of Systems with Infinite State Sets

We now discuss the question whether — for the purpose of control synthesis — finite automata can serve as approximate models for systems with an infinite state set. The motivation for investigating this problem stems from the area of hybrid dynamical systems. There, at least one discrete event system, e.g., in the form of a finite Moore or Mealy automaton, interacts with at least one system with an infinite state space, typically $\mathbb{R}^n$. A composition of infinite and finite state components would result in a hybrid system with state set $\mathbb{R}^n \times X$. This is neither finite nor a vector space, as the finite automaton state set $X$ is typically without mathematical structure. Hence, neither established control synthesis methods from continuous systems

theory nor from the area of supervisory control of *discrete event systems* (DES) can be applied. In this situation, it is then natural to ask whether the infinite state component can be approximated by a suitable finite state automaton, and if control for the resulting overall DES can be meaningful for the underlying hybrid system.

Assume that the component to be approximated can be represented by a state model $P$ defined on a discrete, but not necessarily equidistant, time axis. Assume furthermore that the input is free and that both the input and output spaces are finite. This is natural in the described context, where the input and output signals connect the component's infinite state space to (the) finite DES component(s) in the overall hybrid system (Fig. 1.6).

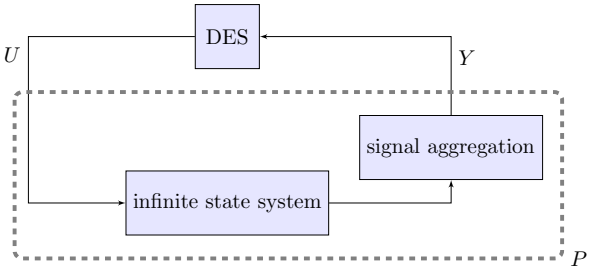


**Fig. 1.6** Hybrid system

Furthermore, we do not assume any knowledge on the initial state. The resulting model is then the following infinite state machine:

$$P = (X, U, Y, f, g, X_0),$$

where $X = \mathbb{R}^n$ is the state space, $U$ and $Y$ are finite input and output sets, $f : X \times U \to X$ is the transition function, $g : X \times U \to Y$ is the output function, and $X_0 = \mathbb{R}^n$ is the set of possible initial states. In total analogy to Section 1.2, the input output relation of $P$ is non-anticipating and its behaviour is

$$\mathscr{B} = \left\{ (u,y) \mid \exists x \in \mathbb{R}^{\mathbb{N}_0} \text{ s.t. } x(k+1) = f(x(k),u(k)), y(k) = g(x(k),u(k)) \ \forall k \in \mathbb{N}_0 \right\}.$$

Note that the above system is time invariant in the sense of [19], as shifting a pair $(u,y) \in \mathscr{B}$ on the time axis will never eliminate it from $\mathscr{B}$. Formally, $\sigma\mathscr{B} \subseteq \mathscr{B}$, where $\sigma$ represents the backward or left shift operator, i.e., $(\sigma(u,y))(k) = (u(k+1), y(k+1))$.

To answer the question whether control synthesis for a finite state approximation can be meaningful for $P$, we obviously need to take into account specifications. For DES and hybrid systems, we often have inclusion-type specifications, i.e., the control task is to guarantee that the closed-loop behaviour is a (nonempty!)

subset of a given specification behaviour, thus ruling out signals that are deemed to be unacceptable. In this case, an obvious requirement for the approximation is that its behaviour contains the behaviour of the system to be approximated. The argument for this is straightforward: as linking feedback control to a plant provides an additional relation between the plant input and output sequences, it restricts the plant behaviour by eliminating certain pairs of input and output sequences. Hence, if there exists a controller that achieves the specifications for the approximation, it will — subject to some technical constraints regarding implementability issues — also do this for the underlying system (for a more formal argument see, e.g., [10, 11]). There is another issue to be taken into consideration, though. This is the question of whether the approximation is sufficiently accurate. In Willems' behavioural framework, e.g. [19, 20], a partial order on the set of all models which relate sequences of symbols from $U$ and $Y$ and which are unfalsified (i.e., not contradicted by available input/output data) is readily established via the $\subseteq$ relation on the set of the corresponding behaviours. In particular, if $\mathscr{B}_A \subseteq \mathscr{B}_B$ holds for two models $A$ and $B$, the former is at least as accurate as the latter. $B$ is then said to be an abstraction of $A$, and, conversely, $A$ is said to be a refinement of $B$. It is well possible that an approximation of the given infinite state model with the required abstraction property is "too coarse" to allow successful controller synthesis. An obvious example is the trivial abstraction, whose behaviour consists of all pairs $(u,y) \in U^{\mathbb{N}_0} \times Y^{\mathbb{N}_0}$. It will allow all conceivable output sequences for any input sequence, and we will therefore not be able to design a controller enforcing any nontrivial specification. Hence, if the chosen approximation of $P$ is too coarse in the sense that no suitable DES controller exists, we need to refine that approximation. Refinability is therefore another important feature we require on top of the abstraction property.

### *1.4.1 l-Complete Approximations*

An obvious candidate for a family of approximations satisfying both the abstraction and refinability property are systems with behaviours

$$\mathscr{B}_l = \left\{ (u,y) \mid (\sigma^t(u,y))\big|_{[0,l]} \in \mathscr{B}\big|_{[0,l]} \ \forall t \in \mathbb{N}_0 \right\}, \ l = 1,2,\ldots \tag{1.4}$$

where $\sigma^t$ is the backward $t$-shift, i.e., $(\sigma^t u)(k) = u(k+t)$, and $(\cdot)\big|_{[0,l]}$ is the restriction operator. The latter maps sequences, e.g., $u \in \mathbb{N}_0$, to finite strings, e.g., $u(0),\ldots,u(l) \in U^{I_l}$. Both the shift and the restriction operator can be trivially extended to pairs and sets of sequences (behaviours). The interpretation of (1.4) is that the behaviour $\mathscr{B}_l$ consists of all pairs of sequences $(u,y)$ that, on any interval of length $l+1$, coincide with a pair of input/output sequences in the underlying system's behaviour $\mathscr{B}$. Clearly,

$$\mathscr{B}_1 \supseteq \mathscr{B}_2 \supseteq \mathscr{B}_3 \supseteq \ldots \supseteq \mathscr{B},$$

i.e., for any $l \in \mathbb{N}$, we have the required abstraction property, and refinement can be implemented by increasing $l$.

A system with behaviour (1.4) is called a *strongest $l$-complete approximation* ([10, 11]) of $P$ : it is an $l$-complete system (e.g., [20]) as checking whether a signal pair $(u,y)$ is in the system behaviour can be done by investigating the pair on intervals $[t,t+l]$, $t \in \mathbb{N}_0$. Formally, a time-invariant system defined on $\mathbb{N}_0$ with behaviour $\tilde{\mathscr{B}}$ is said to be $l$-complete, if $(u,y) \in \tilde{\mathscr{B}} \Leftrightarrow \sigma^t(u,y))\big|_{[0,l]} \in \tilde{\mathscr{B}}\big|_{[0,l]} \ \forall t \in \mathbb{N}_0$. Furthermore, for any $l$-complete system with behaviour $\tilde{\mathscr{B}} \supseteq \mathscr{B}$, it holds that $\tilde{\mathscr{B}} \supseteq \mathscr{B}_l$. In this sense, $\mathscr{B}_l$ represents the most accurate $l$-complete approximation of $P$ exhibiting the abstraction property.

We now need to decide whether an arbitrary pair of input and output strings of length $l+1$, denoted by $(\bar{u},\bar{y})\big|_{[0,l]}$, is an element in $\mathscr{B}\big|_{[0,l]}$, i.e., if the state model $P$ can respond to the input string $\bar{u}\big|_{[0,l]}$ with the output string $\bar{y}\big|_{[0,l]}$. This is the case if and only if $\mathscr{X}((\bar{u},\bar{y})\big|_{[0,l]})$, the set of states of $P$ that are reachable at time $l$ when applying the input string $\bar{u}\big|_{[0,l]}$ and observing the output string $\bar{y}\big|_{[0,l]}$, is nonempty ([11]). $\mathscr{X}((\bar{u},\bar{y})\big|_{[0,l]})$ can be computed iteratively by

$$\begin{aligned}
\mathscr{X}((\bar{u},\bar{y})\big|_{[0,0]}) &= g_{\bar{u}_0}^{-1}(\bar{y}_0)),\\
\mathscr{X}((\bar{u},\bar{y})\big|_{[0,r+1]}) &= f(\mathscr{X}((\bar{u},\bar{y})\big|_{[0,r]}),\bar{u}_r) \cap g_{\bar{u}_{r+1}}^{-1}(\bar{y}_{r+1}), \quad r = 0,\dots l-1,
\end{aligned}$$

where $g_{\bar{u}_r}^{-1}(\bar{y}_r)) := \{\xi \mid g(\xi,\bar{u}_r) = \bar{y}_r\}$ and $f(A,\bar{u}_r) := \{\xi \mid \xi = f(\xi',\bar{u}_r), \xi' \in A\}$.

As $U$ and $Y$ are finite sets, $\mathscr{B}\big|_{[0,l]} = \mathscr{B}_l\big|_{[0,l]}$ is also finite, and a nondeterministic finite Mealy automaton (NDFMeA)

$$P_l = (Z,U,Y,h,Z_0)$$

generating the behaviour $\mathscr{B}_l$ can be set up using the following procedure. It is based on the simple idea that the state of the NDFMeA memorises the past input and output data up to length $l$ , i.e.,

$$z(k) := \begin{cases} \omega & \text{for } \ k = 0,\\ (u(0)\dots u(k-1),y(0)\dots y(k-1)) & \text{for } \ 1 \le k \le l,\\ (u(k-l)\dots u(k-1),y(k-l)\dots y(k-1)) & \text{for } \ k > l, \end{cases}$$

where $\omega$ is a "dummy" symbol meaning "no input/output data recorded so far". Then

$$\begin{aligned}
Z &= \{\omega\} \bigcup_{1 \le r \le l} (U \times Y)^r\\
Z_0 &= \{\omega\}
\end{aligned}$$

and

$$
\begin{aligned}
(\underbrace{(\bar{u}_0,\bar{y}_0)}_{\bar{z}_1},\bar{y}_0) &\in h(\underbrace{\omega}_{\bar{z}_0},\bar{u}_0) \quad \text{iff } (\bar{u}_0,\bar{y}_0)\in\mathscr{B}\big|_{[0,0]}\\
(\underbrace{(\bar{u}_0\dots\bar{u}_r,\bar{y}_0\dots\bar{y}_r)}_{\bar{z}_{r+1}},\bar{y}_r) &\in h(\underbrace{(\bar{u}_0\dots\bar{u}_{r-1},\bar{y}_0\dots\bar{y}_{r-1})}_{\bar{z}_r},\bar{u}_r)\\
&\qquad \text{iff } (\bar{u}_0\dots\bar{u}_r,\bar{y}_0\dots\bar{y}_r)\in\mathscr{B}\big|_{[0,r]},\ 0<r<l\\
(\underbrace{(\bar{u}_1\dots\bar{u}_l,\bar{y}_1\dots\bar{y}_l)}_{\bar{z}_l},\bar{y}_l) &\in h(\underbrace{(\bar{u}_0\dots\bar{u}_{l-1},\bar{y}_0\dots\bar{y}_{l-1})}_{\bar{z}_l},\bar{u}_l)\\
&\qquad \text{iff } (\bar{u}_0\dots\bar{u}_l,\bar{y}_0\dots\bar{y}_l)\in\mathscr{B}\big|_{[0,l]}.
\end{aligned}
$$

### *1.4.2 A Special Case: Strictly Non-anticipating Systems*

It is instructive to briefly investigate the special case where the system to be approximated, $P$, is strictly non-anticipating, i.e., its output function is $g:\mathbb{R}^n\to Y$. This implies that the input $u(k)$ does not affect the output symbols $y(0),\dots,y(k)$. Hence, as the input is free,

$$(\bar{u},\bar{y})\big|_{[0,l]}\in\mathscr{B}\big|_{[0,l]} \text{ iff } \mathscr{X}(\bar{u}\big|_{[0,l-1]},\bar{y}\big|_{[0,l]})\neq\emptyset,$$

where $\mathscr{X}((\bar{u}\big|_{[0,l-1]},\bar{y}\big|_{[0,l]}))$ represents the set of states of $P$ that are reachable at time $l$ when applying the input string $\bar{u}\big|_{[0,l-1]}$ while observing the output string $\bar{y}\big|_{[0,l]}$. As before, we can readily come up with a recursive formula to provide $\mathscr{X}(\bar{u}\big|_{[0,l-1]},\bar{y}\big|_{[0,l]})$:

$$
\begin{aligned}
\mathscr{X}(\bar{y}\big|_{[0,0]}) &= g^{-1}(\bar{y}_0),\\
\mathscr{X}((\bar{u}\big|_{[0,0]},\bar{y}\big|_{[0,1]})) &= f(\mathscr{X}(\bar{y}\big|_{[0,0]}),\bar{u}_0)\cap g^{-1}(\bar{y}_1),\\
\mathscr{X}((\bar{u}\big|_{[0,r]},\bar{y}\big|_{[0,r+1]})) &= f(\mathscr{X}(\bar{u}\big|_{[0,r-1]},\bar{y}\big|_{[0,r]}),\bar{u}_r)\cap g^{-1}(\bar{y}_{r+1}),\ 0<r<l.
\end{aligned}
$$

We can now set up a nondeterministic finite Moore automaton (NDFMoA)

$$\tilde{P}_l=(\tilde{Z},U,Y,\tilde{f},\tilde{g},\tilde{Z}_0)$$

generating the behaviour $\mathscr{B}_l$. This procedure is based on the idea that the automaton state memorises the past input and output data up to length $l-1$ plus the present output symbol, i.e.,

$$
\tilde{z}(k):=\begin{cases}
y(0) & \text{for } k=0,\\
(u(0)\dots u(k-1),y(0)\dots y(k)) & \text{for } 1\le k<l,\\
(u(k-l+1)\dots u(k-1),y(k-l+1)\dots y(k)) & \text{for } k\ge l.
\end{cases}
$$

Hence, for $l > 1$, the state set of the NDFMoA is

$$\tilde{Z} = Y \cup U \times Y^2 \cup \ldots U^{l-1} \times Y^l,$$

and $\tilde{Z}_0 = Y$. The transition function $\tilde{f}$ is defined by

$$\begin{aligned}
\underbrace{(\bar{u}_0, \bar{y}_0\bar{y}_1)}_{\bar{z}_1} &\in \tilde{f}(\underbrace{\bar{y}_0}_{\bar{z}_0}, \bar{u}_0) \quad \text{iff } (\bar{u}_0\%, \bar{y}_0\bar{y}_1) \in \mathscr{B}\big|_{[0,1]} \\
\underbrace{(\bar{u}_0 \ldots \bar{u}_r, \bar{y}_0 \ldots \bar{y}_{r+1})}_{\bar{z}_{r+1}} &\in \tilde{f}(\underbrace{(\bar{u}_0 \ldots \bar{u}_{r-1}, \bar{y}_0 \ldots \bar{y}_r)}_{\bar{z}_r}, \bar{u}_r) \\
&\qquad \text{iff } (\bar{u}_0 \ldots \bar{u}_r\%, \bar{y}_0 \ldots \bar{y}_{r+1}) \in \mathscr{B}\big|_{[0,r+1]}, \quad 1 < r < l-1 \\
\underbrace{(\bar{u}_1 \ldots \bar{u}_{l-1}, \bar{y}_1 \ldots \bar{y}_l)}_{\bar{z}_{l-1}} &\in \tilde{f}(\underbrace{(\bar{u}_0 \ldots \bar{u}_{l-2}, \bar{y}_0 \ldots \bar{y}_{l-1})}_{\bar{z}_{l-1}}, \bar{u}_{l-1}) \\
&\qquad \text{iff } (\bar{u}_0 \ldots \bar{u}_{l-1}\%, \bar{y}_0 \ldots \bar{y}_l) \in \mathscr{B}\big|_{[0,l]},
\end{aligned}$$

where the "don't care" symbol % may represent any element of the input set $U$. For this realisation, the output function is deterministic, i.e., $\tilde{g} : \tilde{Z} \to Y$ and characterised by

$$\begin{aligned}
\tilde{g}(\bar{y}_0) &= \bar{y}_0 \\
\tilde{g}((\bar{u}_0 \ldots \bar{u}_{r-1}, \bar{y}_0 \ldots \bar{y}_r)) &= \bar{y}_r, \quad r = 1, \ldots l-1.
\end{aligned}$$

For $l = 1$, the state only memorises the current output symbol, i.e., $\tilde{Z} = \tilde{Z}_0 = Y$, the transition function $\tilde{f}$ is characterised by

$$\bar{y}_1 \in \tilde{f}(\bar{y}_0, \bar{u}_0) \text{ iff } (\bar{u}_0\%, \bar{y}_0\bar{y}_1) \in \mathscr{B}\big|_{[0,1]},$$

and the output function $\tilde{g}$ is the identity.

**Example 1.11.** We now introduce an example, whose *only* purpose is to illustrate the above procedure. Hence we choose it to be as simple as possible, although most problems will become trivial for this example. It represents a slightly modified version of an example that was first suggested in [15]. Consider the water tank shown in Fig. 1.7. Its cross sectional area is 100cm$^2$, its height $\hat{x} = 30$cm. The attached pump can be switched between two modes: it either feeds water into the tank at a constant rate of 1litre/min, or it removes water from the tank at the same flow rate. The pump is in feed mode if the control input is $u(k) =$ "+", and in removal mode if $u(k) =$ "−". We work with a fixed sampling rate, 1/min, and the control input remains constant between sampling instants. The output signal can take two values: $y(k) = E(mpty)$ if the water level $x(k)$ is less or equal to 15cm, and $y(k) = F(ull)$ if the water level is above 15cm. Hence $U = \{+, -\}$, $Y = \{E, F\}$, and $X = [0, 30\,\text{cm}]$. The behaviour $\mathscr{B}$ can be represented by an (infinite) state model $P = (X, U, Y, f, g, X_0)$, where $X_0 = X$, and $f$ and $g$ are defined by

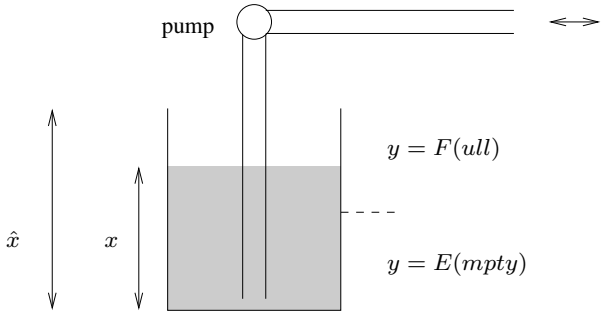


**Fig. 1.7** Simple tank example

$$
\begin{aligned}
x(k+1) &= f(x(k),u(k)) \\
&= \begin{cases} x(k)+10\text{cm} & \text{if } u(k) = \text{“+” and } 0 \le x(k) \le 20\text{cm}, \\ 30\text{cm} & \text{if } u(k) = \text{“+” and } 20\text{cm} < x(k) \le 30\text{cm}, \\ x(k)-10\text{cm} & \text{if } u(k) = \text{“−” and } 10\text{cm} < x(k) \le 30\text{cm}, \\ 0\text{cm} & \text{if } u(k) = \text{“−” and } 0\text{cm} \le x(k) \le 10\text{cm}, \end{cases} \\
y(k) &= g(x(k)) \\
&= \begin{cases} F \text{ if } 15\text{cm} < x(k) \le 30\text{cm}, \\ E \text{ if } 0\text{cm} \le x(k) \le 15\text{cm}. \end{cases}
\end{aligned}
$$

As the system is strictly non-anticipating, we can use the procedure outlined in this section to construct NDFMoA $\tilde{P}_l$ that realise the strongest $l$-complete approximations. Let us first consider the case $l = 1$. We have to check whether strings $(\bar{u},\bar{y})\big|_{[0,1]} \in \mathscr{B}\big|_{[0,1]}$. This is the case if and only if

$$\mathscr{X}(\bar{u}_0,\bar{y}_0\bar{y}_1) = f(g^{-1}(\bar{y}_0),\bar{u}_0) \cap g^{-1}(\bar{y}_1) \neq \emptyset.$$

For our example, we obtain

$$
\begin{aligned}
\mathscr{X}(+,EE) = [10,25] \cap [0,15] = [10,15] \quad &\text{i.e.} \quad (+\%,EE) \in \mathscr{B}\big|_{[0,1]} \\
\mathscr{X}(+,EF) = [10,25] \cap (15,30] = (15,25] \quad &\text{i.e.} \quad (+\%,EF) \in \mathscr{B}\big|_{[0,1]} \\
\mathscr{X}(+,FF) = (25,30] \cap (15,30] = (25,30] \quad &\text{i.e.} \quad (+\%,FF) \in \mathscr{B}\big|_{[0,1]} \\
\mathscr{X}(+,FE) = (25,30] \cap [0,15] = \emptyset \quad &\text{i.e.} \quad (+\%,FE) \notin \mathscr{B}\big|_{[0,1]} \\
\mathscr{X}(-,EE) = [0,5] \cap [0,15] = [0,5] \quad &\text{i.e.} \quad (-\%,EE) \in \mathscr{B}\big|_{[0,1]} \\
\mathscr{X}(-,EF) = [0,5] \cap (15,30] = \emptyset \quad &\text{i.e.} \quad (-\%,EF) \notin \mathscr{B}\big|_{[0,1]} \\
\mathscr{X}(-,FF) = (5,20] \cap (15,30] = (15,20] \quad &\text{i.e.} \quad (-\%,FF) \in \mathscr{B}\big|_{[0,1]} \\
\mathscr{X}(-,FE) = (5,20] \cap [0,15] = (5,15] \quad &\text{i.e.} \quad (-\%,FE) \in \mathscr{B}\big|_{[0,1]}.
\end{aligned}
$$

The described realisation procedure then provides the NDFMoA $\tilde{P}_1$ shown in Fig. 1.8, where the output symbols associated to the states are indicated by dashed arrows. The (initial) state set is $\tilde{Z} = \tilde{Z}_0 = Y$, and there are six transitions. Clearly, $\mathscr{B}$ is a strict subset of the behaviour generated by $\tilde{P}_1$, i.e., $\mathscr{B} \subset \mathscr{B}_1$. For example, $\tilde{P}_1$ allows the sequence $EEE\ldots$ as a possible response to the input sequence $+++\ldots$, i.e., we can add water to the tank for an arbitrary period of time without ever observing the output symbol $F$. This would clearly not be possible for the system to be approximated.

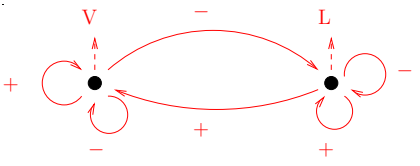


**Fig. 1.8** Realisation $\tilde{P}_1$ of strongest 1-complete approximation

We now proceed to the case $l = 2$. To construct the strongest 2-complete approximation, we need to establish whether strings $(\bar{u},\bar{y})\big|_{[0,2]} \in \mathscr{B}\big|_{[0,2]}$. As stated above, this is true if and only if

$$\mathscr{X}(\bar{u}_0\bar{u}_1,\bar{y}_0\bar{y}_1\bar{y}_2) = f\left(f(g^{-1}(\bar{y}_0),\bar{u}_0)\cap g^{-1}(\bar{y}_1),\bar{u}_1\right)\cap g^{-1}(\bar{y}_2) \neq \emptyset.$$

For example, we obtain

$$\mathscr{X}(++,EEF) = f\left(\underbrace{f(g^{-1}(E),+)\cap g^{-1}(E)}_{\mathscr{X}(+,EE)},+\right)\cap g^{-1}(F)$$
$$= [20,25]\cap(15,30] \neq \emptyset \quad \text{i.e.} \quad (++\%,EEF) \in \mathscr{B}\big|_{[0,2]}.$$

Repeating this exercise for other strings and using the realisation procedure described above, we obtain the NDFMoA $\tilde{P}_2$ shown in Fig. 1.9. Its state set is $\tilde{Z} = Y \cup U \times Y^2$, its initial state set $\tilde{Z}_0 = Y$. Note that only 8 out of the 10 elements of $\tilde{Z}$ are reachable. These are the initial states and states $(\bar{u}_0,\bar{y}_0\bar{y}_1)$ such that $(\bar{u}_0\%,\bar{y}_0\bar{y}_1) \in \mathscr{B}\big|_{[0,1]}$. To avoid unnecessary cluttering of the figure, the output symbols are not indicated for each state, but summarily for all states generating the same output.

We can readily check that for this simple example $\mathscr{B}_2 = \mathscr{B}$, i.e., the NDFMoA $\tilde{P}_2$ exhibits exactly the same behaviour as the underlying infinite state model $P$. In general, however, no matter how large $l$ is chosen, we cannot expect that the behaviour generated by $P$ is $l$-complete and, in consequence, we will *not* be able to recover it exactly by an $l$-complete approximation. ■

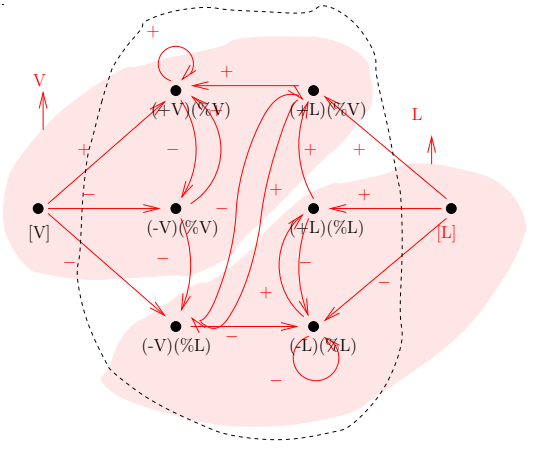


**Fig. 1.9** Realisation $\tilde{P}_2$ of strongest 2-complete approximation

**Remark 1.12.** Suppose we design a feedback controller, or supervisor, for the approximating automaton $P_l$ or $\tilde{P}_l$. Clearly, this controller must satisfy the following requirements: (i) it respects the input/output structure of $P_l$ (respectively $\tilde{P}_l$), i.e., it cannot directly affect the output $y$; (ii) it enforces the (inclusion-type) specifications when connected to the approximation, i.e., $\mathscr{B}_l \cap \mathfrak{B}_{\text{sup}} \subseteq \mathfrak{B}_{\text{spec}}$, where $\mathfrak{B}_{\text{sup}}$ and $\mathfrak{B}_{\text{spec}}$ are the supervisor and specification behaviours, respectively, and where we assume that $\mathfrak{B}_{\text{spec}}$ can be realised by a finite automaton; (iii) the approximation and supervisor behaviours are nonconflicting, i.e., $\mathscr{B}_l\big|_{[0,k]} \cap \mathfrak{B}_{\text{sup}}\big|_{[0,k]} = (\mathscr{B}_l \cap \mathfrak{B}_{\text{sup}})\big|_{[0,k]}$ for all $k \in \mathbb{N}_0$, i.e., at any instant of time, the approximation and the controller can agree on a common future evolution. In this chapter, we will not discuss the solution of this control synthesis problem. [11] decribes how the problem can be rewritten to fit the standard supervisory control framework (e.g., [16, 17]), and Chapter 3 of this book discusses how to find the minimally restrictive solution to the resulting standard problem. If we find a nontrivial (i.e., $\mathfrak{B}_{\text{sup}} \neq \emptyset$) controller for the approximation, we would of course like to guarantee that it is also a valid controller for the underlying infinite state system $P$, i.e., items (i), (ii), and (iii) hold for $P$ and its behaviour $\mathscr{B}$. As $P$ and $P_l$ (respectively $\tilde{P}_l$) exhibit the same input/output structure, (i) is straightforward and (ii) follows immediately from the abstraction property of $P_l$ (respectively $\tilde{P}_l$). There is another (not very restrictive) technical requirement that ensures that (iii) also holds for $\mathscr{B}$, see [10, 11] for details. ■

**Remark 1.13.** From the construction of $P_l$ (respectively $\tilde{P}_l$), it is immediately clear that the order of the cardinality of the approximation state set is exponential in the parameter $l$. In practice, one would therefore begin with the "least accurate" approximation $P_1$ (respectively $\tilde{P}_1$) and check whether a nontrivial control solution can be found. If this is not the case, one turns to the refined approximation $P_2$ (respectively $\tilde{P}_2$). In this way, refinement and control synthesis alternate until either a nontrivial

control solution is found or computational resources are exhausted. Hence, failure of the control synthesis process to return a nontrivial solution triggers a "global" refinement step, although "local" refinements could well suffice. A more "intelligent" procedure suggested in [12] therefore analyses failure of the synthesis process and focuses its efforts on those aspects of the approximation that have "caused" the failure in synthesis. This procedure "learns from failure" in the synthesis step and thus implements a refinement that is tailored to the particular combination of plant and specification. ■

## 1.5 Further Reading

Finite automata, including Mealy and Moore automata are discussed in a vast number of standard textbooks. Examples are [3, 7]. The latter contains a number of instructive examples of how Moore and Mealy automata model various systems of practical interests. [4] is *the* standard textbook on discrete events systems, and also includes a number of examples that illustrate how finite generators can be used to model engineering problems.

The problem of approximating systems with infinite state space by finite state machines has attracted a lot of attention since hybrid systems theory became "en vogue". There are various approaches, and the reader is advised to consult special journal issues on hybid systems, e.g., [2, 5, 13], proceedings volumes such as [1, 9], or the recently published survey book [8] for an overview. In this chapter, we have concentrated on finding approximations with the so-called abstraction property. The closely related concepts of simulations and approximate bisimulations are discussed, e.g., in [6, 18].

## References

1. Antsaklis, P.J., Kohn, W., Lemmon, M.D., Nerode, A., Sastry, S.S. (eds.): HS 1997. LNCS, vol. 1567. Springer, Heidelberg (1999)
2. Antsaklis, P.J., Nerode, A.: IEEE Transactions on Automatic Control–Special Issue on Hybrid Control Systems 43 (1998)
3. Booth, T.L.: Sequential Machines and Automata Theory. John Wiley, New York (1967)
4. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Kluwer Academic Publishers, Boston (1999)
5. Evans, R., Savkin, A.V.: Systems and Control Letters–Special Issue on Hybrid Control Systems 38 (1999)
6. Girard, A., Pola, G., Tabuada, P.: Approximately bisimilar symbolic models for incrementally stable switched systems. IEEE Transactions on Automatic Control 55(1), 116–126 (2010)
7. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)

8. Lunze, J., Lamnabhi-Lagarrigue, F. (eds.): The HYCON Handbook of Hybrid Systems Control: Theory, Tools, Applications. Cambridge University Press (2009)
9. Lynch, N.A., Krogh, B.H. (eds.): HSCC 2000. LNCS, vol. 1790. Springer, Heidelberg (2000)
10. Moor, T., Raisch, J.: Supervisory control of hybrid systems within a behavioural framework. Systems and Control Letters–Special Issue on Hybrid Control Systems 38, 157–166 (1999)
11. Moor, T., Raisch, J., O'Young, S.: Discrete Supervisory control of hybrid systems by l-Complete approximations. Journal of Discrete Event Dynamic Systems 12, 83–107 (2002)
12. Moor, T., Davoren, J.M., Raisch, J.: Learning by doing – Systematic abstraction refinement for hybrid control synthesis. IEE Proc. Control Theory & Applications–Special Issue on Hybrid Systems 153, 591–599 (2006)
13. Morse, A.S., Pantelides, C.C., Sastry, S.S., Schumacher, J.M.: Automatica–Special issue on Hybrid Control Systems 35 (1999)
14. Raisch, J., O'Young, S.D.: Discrete approximation and supervisory control of continuous systems. IEEE Transactions on Automatic Control-Special Issue on Hybrid Systems 43(4), 569–573 (1998)
15. Raisch, J.: Discrete abstractions of continuous systems–an Input/Output point of view. Mathematical and Computer Modelling of Dynamical Systems–Special Issue on Discrete Event Models of Continuous Systems 6, 6–29 (2000)
16. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event systems. SIAM Journal of Control and Optimization 25, 206–230 (1987)
17. Ramadge, P.J., Wonham, W.M.: The control of discrete event systems. Proceedings of the IEEE 77, 81–98 (1989)
18. Tabuada, P.: Verification and Control of Hybrid Systems: A Symbolic Approach. Springer-Verlag, New York Inc. (2009)
19. Willems, J.C.: Models for Dynamics. Dynamics Reported 2, 172–269 (1989)
20. Willems, J.C.: Paradigms and puzzles in the theory of dynamic systems. IEEE Transactions on Automatic Control 36(3), 258–294 (1991)
21. Wonham, W.M.: Notes on control of discrete event systems. Department of Electrical & Computer Engineering. University of Toronto, Canada (2001)